Real World Software Engineering

Final Report

Donald Gotterbarn, Robert Riser, and Suzanne Smith

July 15, 1994

U.S. Army Research Office

Contract/Grant DAAL03-92-G-0411

East Tennessee State University

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** 6/31/94	**3. REPORT TYPE AND DATES COVERED** Final Report: 28 Sep 92-31 May 94

4. TITLE AND SUBTITLE

Real World Software Engineering

5. FUNDING NUMBERS

DAAL03-92-G-0411

6. AUTHOR(S)

Donald Gotterbarn
Robert Riser
Suzanne Smith

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

East Tennessee State University
Department of Computer and Information Sciences
Box 70711
Johnson City, TN 37614-0711

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ARO 30996.4-MA

11. SUPPLEMENTARY NOTES

The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Real World Software Engineering project involved the development and implementation of a two-semester undergraduate software engineering course which provides thorough coverage of the software development process along with realistic and varied project experiences. The course is built around three projects which differ in several significant ways: size, complexity, team structure, deliverables, and development methodology. The projects are carefully choreographed to provide varied team experiences and allow students to function in a variety of roles and responsibilities.

Coordinated lecture, laboratory, and project activities are provided. A layered approach in which topics are initially introduced and revisited in increasing depth is utilized in lectures and project work. Ada is used first as a specification and design tool and later as an implementation language. Continuous assessment with an emphasis on reviews is utilized in the projects.

Deliverables of the project include an overview of the course, detailed syllabus, an integrated schedule of lecture, laboratory, and project activities, lectures and associated handouts, laboratories and associated handouts, methods & materials related to managing and assessing projects.

14. SUBJECT TERMS

Software engineering education

15. NUMBER OF PAGES 1007

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Broad Agency Announcement (BAA) # 92-25

Category 2

SECTION A

Real World Software Engineering

Points-of-Contact

Technical:

Professors Donald Gotterbarn,
and Robert Riser

East Tennessee State University
Department of Computer
and Information Sciences
Box 70,711
Johnson City, Tennessee 37614-0711
(615) 929-6849,5609
I01gbarn@etsu,I01riser@etsu
FAX: (615) 461-7119

Administrative:

Professor Gordon Bailes
East Tennessee State University
Department of Computer
and Information Sciences
Box 70,711
Johnson City, Tennessee
37614-0711
(615) 929-6958
I01baile@etsu.bitnet

Structure of the deliverable

Real-World Software Engineering

I INTRODUCTION

a. Purpose and Goals

Based on our experience teaching software engineering, we at East Tennessee State University are convinced that a one-semester software engineering course cannot adequately cover all aspects of the software development process and still provide students with meaningful project experience. Current software engineering course models emphasize either the product or the process [Shaw 91]. These models rarely finish a realistic product or do so by marginal treatment of significant aspects of the life cycle. For example, while concentrating on implementation details, topics such as detailed design reviews, configuration management, and maintenance are minimized.

To address this problem, East Tennessee State University is expanding and changing its undergraduate curriculum in software engineering. Integral to this effort, we are incorporating into the undergraduate curriculum lessons learned while developing and teaching software engineering courses at the graduate level. This proposal was to develop a two-semester undergraduate course which presents real-world software engineering. The course provides a thorough coverage of the software development process with realistic project experience.

The course is designed to present software engineering in a layered approach where "inter-related topics are presented repeatedly in increasing depth" [Ford 87]. Furthermore, the relationship of software engineering principles to software development is emphasized by the careful coordination of project and lecture stages [Shaw 91]. For example, in the first four weeks the students are rapidly introduced to the fundamental principles of software engineering concurrent with a small project. During the remainder of the first semester, a thorough examination of analysis and design and their controlling disciplines is presented. The second semester addresses the remaining principles of a complete, mature software development process [Humphrey 88].

In order to provide an instructional mechanism and realistic project experience, the course uses both the "small group project" model and the "large project team" model [Shaw 91]. The first project is a "toy project" which is fully specified by the instructor. The management organization for this project is a chief programmer team [Brooks 82]. During this four-week project, the students are also introduced to Ada using a "program reading methodology" [Deimel 90]. Ada is also used as the specification language. As the toy project nears completion, a large project for an external client is introduced. A matrix management organization is used for this project. The first semester takes this project through Preliminary Design Review. Successful deliverables from this semester will be used in the second semester. Emphasis is placed on validation techniques for requirements and design. CASE tools are used to document and validate the designs.

The second semester has three major project components. First is the completion of a large project involving a real client. This project begins with a baselined design document, specified in Ada, and continues through acceptance testing. The second major component is multiple small maintenance requests applied to an Ada artifact. This disciplined approach to maintenance gives the students experience needed by industry, which is rarely achieved in traditional software engineering courses. Finally, during a four-week assessment period, various formal methods, metrics, and tools are applied to the three course projects. In this assessment, both the processes and the products are evaluated to capture their strengths and weaknesses.

Innovations include:

* Ada - first introduced through program reading, then as specification and design notation, later as an implementation language;

* Industrial Setting - working on multiple teams and team organizations, working on different sizes and types of projects, assuming different roles, experience with a variety of CASE tools;

* Continuous Assessment - integrated into all project activities using formal reviews with an emphasis on validation and verification throughout the life cycle;

* Closing Assessment Period - a period dedicated to appraising the strengths and weaknesses of the processes and products discussed and developed during the course; and

* Professionalism - integration of professional, ethical, and legal issues in accordance with the recommendations of the IEEE/ACM Computer Society Task Force.

b. Technical details relating to this document

All of the materials used in this course have been formatted in WordPerfect 6.0 for windows. The entire document can be printed using that software. We have provided it in this format to enable easy modification and adaptation by anyone using these materials for teaching software engineering. The only restriction is on the material in section d below which is from a copyrighted paper. This entire package of materials is also available in postscript format. Both the wordperfect and postscript versions of this material are availabl϶ from the Defense Documentation Center..

c. Structure of the document.

These materials are designed to be used as a complete two semester course in software engineering. The materials contained herein can be used either in whole of in part. For example, most of the projects can be adapted to a one semester course in software engineering, or a course which uses C++, rather than Ada as the language of choice. The lectures are self contained, all relevant overheads and handouts are associated with each lecture. The document is divided up by pedagogical tasks associated with teaching an extended software engineering course. The tasks include lectures, development and management of software development projects, assessing the students work in those projects and their understanding of the course materials, and tutoring them in the additional languages required to do their projects. Although the work is divided into sections dealing with each of these tasks, the material in each of these sections is cross referenced to relevant material in other sections. We have also included a section on resources available in the summer of 1994.

d. Course Overview

In the development of this course, we presented our research in several forums, including the Seventh Conference on Software Engineering Education. The description of the course presented there is appended below.

Real-World Software Engineering:

A Spiral Approach to a Project Oriented Course [1]

Donald Gotterbarn and Robert Riser

East Tennessee State University
Johnson City, Tennessee 37604-0711

Abstract. A one-semester course cannot adequately cover the software development process and still provide meaningful project experience. We have developed and implemented a tightly- coupled two-semester undergraduate course which presents, in a *spiral form*, theory and practice, product and process. Coordinating the increase in depth of the lectures as topics are revisited repeatedly, with increasingly demanding projects, constitutes our spiral approach. Three projects differ in size, complexity, team structure, artifacts provided and delivered, and development methodologies. The projects are carefully choreographed to provide varied team experiences and allow each student to function in a variety of roles and responsibilities. The project framework provides a series of passes through the software development process, each pass adding to a body of common student experiences to which subsequent passes can refer. By the middle of the first semester students, individually and in teams, have begun accumulating their own "war stories"; some positive, some negative. This personalized knowledge provides a solid base for more advanced concepts and classroom discussion.

1 Introduction

Based on our experience teaching software engineering, we are convinced that a one-semester software engineering course cannot adequately cover all aspects of the software development process and still provide students with meaningful project experience. Current software engineering course models emphasize either the product or the process [Shaw 91]. These models rarely finish a realistic product or do so by marginal treatment of significant aspects of the life cycle and premature immersion in implementation details. For example, while concentrating on implementation details, topics such as detailed design reviews, configuration management, and maintenance are not given adequate attention.

[1] **This project was partially funded by DARPA research grant DAAL03-92-G-0411.**

To address this problem, we have expanded and changed our undergraduate curriculum in software engineering. Integral to this effort we have incorporated lessons learned while developing and teaching software engineering courses at the graduate level. Moreover, we integrate graduate software engineering milestone reviews into the undergraduate software engineering classroom. A DARPA grant enabled us to complete development and implementation of a two-semester undergraduate course which presents, in a spiral form, theory and practice, product and process, throughout the tightly coupled two-semesters; mimicking a real-world software engineering process. [2]

Our course differs from other multi-semester courses in two ways. First, rather than separating theory and practice into different semesters [Adams 93]; we blend them throughout. Second, rather than mistakenly presenting the software development life cycle as two discrete pieces, analysis and design in one semester and code and test in the other, we more accurately model the iterative nature of software development. Our approach combines a thorough coverage of the software development process with realistic project experience.

This paper describes our course, related experiences, and lessons learned during its development and initial offerings.

2 The Approach

The two-course sequence is designed to present software engineering in a layered approach where "inter-related topics are presented repeatedly in increasing depth" [Ford 87]. Furthermore, the relationship of software engineering principles to software development is emphasized by the careful coordination of project and lecture stages [Shaw 91]. The combination of these two techniques, coordinating the increase in depth of the lectures with more demanding project experiences, constitutes our spiral approach.

The course is built around three projects which differ in several significant ways: size, complexity, team structure, artifacts provided and delivered, and development methodologies. The projects are carefully choreographed to provide varied team experiences and allow each student to function in a variety of roles and responsibilities.

In the first five weeks the students are rapidly introduced to the fundamental principles of software engineering and, while working in teams, they complete a modest development project. Despite the introduction of sound software engineering principles, the simplicity of the project allows student teams to concentrate on the end product rather than the development process and still achieve a modicum of success.

As the first project nears completion, a second, extended project with a real customer is introduced. It spans both semesters and requires revisiting concepts in depth that were merely touched upon in the first project. The large project is also a vehicle to introduce

[2] **The syllabus for the course is included as Appendix A.**

and utilize new concepts, such as detailed design and configuration management. The use of a real customer provides an opportunity to study more complex requirements and exposes students to problems which were not apparent in the small project. The added complexity, introduced by size, real customer, and intricate requirements, demands the use of more effective controlling disciplines and increased attention to the software process.

The third project requires the students to perform maintenance on an existing large software system. To mimic the typical industrial situation, these maintenance tasks are assigned while the students are still working on the large project. Work on the maintenance tasks and the large project overlap and they have a common due date. These tasks provide yet another opportunity to revisit and reinforce significant software engineering concepts, but this time from a maintenance perspective. Maintenance is treated as a complete software development task. Students can now understand the benefits of following good software engineering practices.

Finally, during a four-week assessment period, various formal methods, metrics, and tools are applied to the three course projects. In this assessment, both the processes and the products are evaluated to capture their strengths and weaknesses.

3 The Projects

In order to provide an instructional mechanism and realistic project experience we combine two models from [Shaw 91], the "small group project" and the "large project team" and supplement this with a set of maintenance tasks and a closing assessment period.

3.1 Project 1: The Modest/Toy Project

The requirements are provided and students are expected to specify, design, code, and test a solution. Toy projects recently used included a bottle and can recycling device, an automated fire and security alarm system, a kiosk vending machine system and an EMS-911 telephone exchange.[3] The toy project is scheduled for weeks 2 through 6 of the first semester. Since work must begin quickly, controlling disciplines are imposed upon the teams with minimum justification at this point. For example, students are immediately introduced to various lifecycle elements (scheduling, project organization, configuration management, quality assurance, and verification and validation techniques) by "living them" but only later are these topics formally addressed in lectures. While the project is implemented in a language familiar to the student, Ada specifications are introduced in high-level design.

These projects involve minimal logical complexity so that the students might devote their

[3] See Appendix B for some examples of such projects.

attention to the details of the design and development of the software. Students are asked to mimic a waterfall lifecycle. Teams are limited to four to six members each and we have found that instructors can successfully manage up to three different toy projects. Keeping track of the details of more than three simultaneous projects imposes a considerable burden without any benefit. Of course this means that for larger classes multiple teams will be working independently on the same project. There are some educational benefits to having several teams attempting the same project.

A democratic team organization is used for toy projects. At this point in the course, the instructor has inadequate knowledge of individual student's project-oriented skills to be able to place them in other organizational models. Because each project is relatively small, students approach it as individuals in an *ad hoc* fashion. Careful professorial management is required to minimize this mistake. As a means of tracking progress and focusing their efforts, a software project management plan (SPMP)[4], including scheduled product reviews and deliverables, is provided. This software project management plan applies equally well to all of the toy projects.

Due to time constraints, we strongly recommend that the professor serve as the user for these projects. As the user, the instructor must assume a naivete about computing and only answer questions from the user perspective. We have found that it is helpful to declare which role --customer or professor-- is being assumed at any given time (e.g. during requirements clarification and formal reviews of deliverables). This is necessary to resist "professorial micro management" of projects. This over-management problem is further minimized by the involvement of other faculty in roles such as user, customer, staff, and reviewer.

In order to encourage meeting deadlines, we require regular team meetings. To help students who have not experienced task oriented meetings, we provide a task oriented team meeting report form[5]. We use this form to describe how to control and track tasks. Completed team meeting reports are required at the beginning of each week and any common problems are discussed with the class. Later this material is revisited in discussions of project status reports and assessment techniques.

The team meeting reports also serve as an early warning system for a variety of personnel problems. Even at this early stage, students sometimes shirk their responsibilities. We recommend team sizes of six students: if one or two students fail to contribute, or leave the team, it can still successfully function.

During this project, students are introduced to Ada through a "program reading methodology" [Deimel 90] using several artifacts developed especially for the course. At this point Ada is used only as a specification language. We have found John Herro's

[4] **A sample modest project management plan is contained in Appendix C.**

[5] **A sample team meeting report form is contained in Appendix D.**

shareware tutorial, The Interactive Ada Tutor[6], to be useful as a self-paced introduction to Ada. As the toy project nears completion, a large project for a real customer is introduced.

The deliverables from each team's project include a requirements analysis document, a system design, the outline of a test plan which is traceable to the requirements, test cases, meeting reports, and an implemented system.

3.2 Project 2: The Extended Project

Beginning with an initial request from a "real customer", students are expected to complete all aspects of a solution, from requirements engineering (elicitation, analysis, and specification) through implementation. This project begins in week five of the first semester and extends through week eleven of the second semester. Analysis and design, up through Ada specifications, are completed by the end of the first semester with detailed design, coding, and testing to follow in the second semester.

Several items introduced in project one are revisited and expanded upon here, including reviews, controlling techniques, software development standards, Ada as a software development tool, and development team organizations.

Internal project reviews are emphasized [Bruegge 91]. The SPMP for the extended project[7] requires reviews at appropriate places. For example, students experience for the first time a formal requirements review in the presence of a customer. The schedule includes time for them to modify their documents based on the reviews. Students are uncomfortable reviewing the work of their peers and uncomfortable presenting their work to peers. We address these two problems in several ways. The reviews are highly structured by providing the students with general guidelines for a review process and specifications for the content of preliminary and detailed designs[8]. We have found that it also helps to have another faculty member, who is carefully coached to assume an attitude of constructive criticism, participate in the reviews.

In some cases we have multiple reviews on the same day for teams which have an obvious vested interest in the other team's work. This interest, even if generated out of self-defense, guarantees careful prior attention to the material being reviewed. For example, the preliminary user manual review and the preliminary requirements review are scheduled for the same day. These reviews also provide ample opportunity for "planned spontaneity" on the part of the instructor. The multiple review approach insures that other viewpoints are heard and prompts an apparently spontaneous discussion of viewpoint analysis and resolution.

[6] Useful introductory Ada tools include: [Herro 88], [Benjamin 91], and [Booch93].

[7] A sample project management plan for extended projects is in Appendix E.

[8] The review guidelines and formats for detailed and preliminary design are contained in Appendix F.

A tool to help students overcome their concerns with reviews is an educational materials package from the Software Engineering Institute. The package includes a video-tape "Scenes of Software Inspections" and discussion aids. [Deimel 91] In less than 20 minutes, students see several dramatizations of common pitfalls in formal reviews. The presentation makes the pitfalls and the problems they generate obvious to the students. Each dramatization is intended to be followed by a discussion of how to avoid these pitfalls. This discussion reduces anxiety about reviews and develops an appreciation of appropriate review roles and behavior. The students are required to attend at least one formal review in our graduate software engineering program.

While Ada was introduced in the high level specification of project one, it is now used as a requirements specification and design tool. It is also the implementation language for the extended project. Following our spiral approach, the program reading methodology is continued. The examples and classroom exercises provided go into greater depth. In-class discussion of Ada syntax is minimized and there is a continued reliance on self-paced Ada tutorial materials, laboratory experiences, and the Ada Language Reference Manual [ANSI/MIL-STD-1815a, 1983].

We now justify the controls which were imposed in the toy project. Recognized standards, such as DOD, NASA, or IEEE models, are formally introduced and are required for all project documents and procedures. The size and complexity of the current project helps students appreciate the importance of all aspects of the standards, both managerial and technical, in controlling both process and product. The use of accepted controlling techniques is also reinforced.

The project team organization changes dramatically for this project. Rather than multiple projects with democratic teams, the entire class is organized to work on a single project and students assume roles on various functional teams, e.g., requirements, configuration management, testing, design, and programming. Several of these teams start work immediately following the client request.

New concepts are also introduced in project two, including rigorous controlling techniques such as configuration management, formal test plans, team walkthroughs and inspections, a matrix organization requiring inter-team and intra-team communication, verification and validation, software quality assurance, and requirements elicitation.

Configuration management is enforced. A configuration management plan is developed by a student selected as configuration manager. This plan is developed and presented to the class for review.[9] The revised plan is automated and in place prior to the development or submission of any other configuration items. From this point on, all documents submitted for formal review are immediately placed under configuration management and subsequent modifications must follow the configuration management plan.

The careful selection of a configuration manager (CM) greatly improves the chances for a successful project. The student selected as the CM is placed in a unique position among

[9] A sample plan developed by a student is attached as Appendix G.

peers. The instructor, like other managers, must provide appropriate support and direction for the CM.

Because students have little exposure to formal test design and testing methods, we provide them with a sample test plan. Because the sample test plan is keyed to requirements and design, we use it to introduce traceability. Most students view testing simply as code verification. To address this narrow view we require that the test team begin work on its plan shortly after requirements analysis is underway. The degree of abstraction of the requirements forces the test team to treat testing as a complete lifecycle issue.

In addition to revisiting formal reviews, we add required team inspections and walkthroughs of their configuration items. These processes occur during team meetings. To give the widest possible range of experiences, the students are required to function in two different review roles on each team during the semester. Since each student is on two teams, they experience four different roles.

A significant aspect of this project is our employment of a matrix organization. The class is organized as a project team working on a single project. This resembles a functional organization. We make it resemble a matrix organization when we divide the class into several functional teams, as described above. Each student, with the exception of the CM, serves on at least two teams [Stuckenbruck 81]. The correct allocation of students to functional teams is critical for project integrity. For example, students should not be on both the coding and the test team. A critical guideline is that no student be assigned to two teams which are responsible for validating each other's work. Many teams act as cross checks on each other during development. For example, if at all possible there should be a user's manual team which meets independently with the user, while the requirements team meets with the customer. During the requirements review the user's manual team can be used to help validate the requirements. Appendix H contains a model of a matrix organization for a class of fifteen students; a model for a class of twenty-five students has also been developed.

This methodology has the virtue of placing many students in leadership roles. Because teams which must communicate directly with each other have no common students, a higher level of precision is required in inter-team communications. They cannot rely on a student who is on both the sending and the receiving team to clarify document ambiguities. Although most students function as members of only two teams, they learn about the functions and products of the other teams through the review process.

The first semester takes this project through preliminary design review. Emphasis is placed on validation techniques for requirements and design. Successful deliverables, specified in Ada, from this semester become baseline documents for the second semester. The second semester begins detailed design and continues through acceptance testing. The deliverables from the class and teams have included: requirements documents from the requirements team; test plan and testing report from the test team; configuration management plan, change report log, system build report from the configuration manager; preliminary and detailed design documents from the respective design teams; meeting reports from all teams, and an implemented and accepted system.

3.3 Project 3: The Maintenance Project

Another major component of the second semester involves multiple maintenance requests applied to a large Ada artifact. This disciplined approach to maintenance gives the students experience needed by industry but rarely achieved in traditional software engineering courses.

Students perform major maintenance (including corrective, enhancement, and adaptive activities) on an existing software system. A maintenance configuration management plan which introduces version control techniques, and a maintenance project management plan is provided. The maintenance project is scheduled for weeks six through eleven of the second semester, overlapping the extended project. A variety of maintenance tasks, like those described by Engle, Ford, and Korson [Engle 89], are assigned. Without guidance students tend to revert to "code and fix" habits.

A new project organization is introduced here. The students are organized into chief-programmer teams [Brooks 82]. The choice of chief-programmer is based on our knowledge of the students' skills and attitudes demonstrated on the other course projects. Each team is given responsibility for different maintenance tasks [Callis 91]. These tasks, applied to a single large artifact, require inter-team communication and stronger change control, and introduce the problem of maintaining conceptual integrity. This new complexity provides new challenges to the student CM.

The maintenance project helps students see the utility of controlling techniques during original development. By equating maintenance and development the students revisit most of the concepts previously discussed. This third trip through the spiral makes it easier for them to work with a large unfamiliar artifact. Many students find this somewhat surprising and rewarding.

3.4 Project Assessment Period

Continuous assessment is integrated into all project activities using formal reviews and an emphasis on validation and verification throughout the life cycle. In addition, an extended closing assessment period is dedicated to appraising the strengths and weaknesses of the processes and products discussed and developed during the course.

This assessment period, based on the final phase of the Design Studio course from Carnegie Mellon's Master of Software Engineering curriculum [Tomayko 91], takes place during the final four weeks of the second semester. It also incorporates aspects of the lessons learned document of the NASA software development standard [NASA 86]. Students learn to be constructively critical of their own work and to be realistic about their plans. The major purpose is to determine to what degree the original project plans were realized and to discover shortcomings of the software product and, perhaps more importantly, the software process. The assessment includes an analysis of possible product improvements and a discussion of how to revise the product accordingly.

4 Innovations and Advantages of this Approach

This course provides a commercial-like environment where students work on multiple teams and team organizations, work on multiple projects, and assume different roles. This interplay of models accurately reflects what the students will encounter in industry. This setting is also modeled by using a variety of project types, namely, the "real client" and the "toy project" described by Bruegge, Cheng, and Shaw [Bruegge 91]. Our projects collectively meet the standards set forth by Shaw and Tomayko. For example, the large project has a real customer and a target audience. "A project with a real client is the best motivator" [Shaw 91]. But this project is only pursued after the students have completed a smaller project and have been exposed to the proper techniques of software development. Students will gain programming-in-the-large experiences on the extended project and on the maintenance project. Acquisition of new domain knowledge, another standard set forth by Shaw and Tomayko, is required to some extent in all three projects. Finally, configuration management tools appropriate to each type and size of project are used [Shaw 91]. These projects provide both a teaching mechanism and realistic project experience for the students.

Multiple modes of communication are experienced. The democratic model gives the students experience with a small project and intra-team communication. The matrix organization gives the students experience with inter-team communication. The maintenance project requires both of these forms of communication. All of these forms of communication are needed by the successful software engineer.

ETSU's College of Applied Science and Technology has an ongoing emphasis on written and oral communication skills. In all work for this course, including reviews, formal presentations and documents, the students are required to adhere to the standards as specified in the *Language Skills Handbook* [AST 90]. Reviews and presentations could be videotaped for review, development and evaluation.

Ada is used throughout all course activities. This is our students' first exposure to Ada. It is introduced early in the first semester using program reading techniques [Deimel 90]. For example, students learn to read Ada specifications as illustrations of simple designs. At the same time, Ada's complexities are progressively introduced by reading other Ada examples. In addition to program reading and extensive use of Ada examples, students learn to write high-level design specifications in Ada. A major objective is to have the students produce and validate a complete Ada specification of a large project by the end of the first semester. Students come to view Ada as more than an implementation language. During the second semester, the large project is implemented in Ada, and maintenance is performed on an existing Ada software system. We use *Ada Quality and Style: Guidelines for Professional Programmers* [SPC 91] as our Ada style guide.

Professional, ethical, and legal issues are integrated into both the lecture and laboratory components of the course. This is consistent with the recommendations of the IEEE/ACM Computer Society Task Force. Our model of the industrial setting provides a context in which to discuss a range of ethical situations not normally encountered in typical software engineering courses.

5 Conclusion

We have found this spiral approach to be an effective teaching and learning tool. The project framework provides a series of passes through the software development process, each pass adding to a body of common student experiences to which subsequent passes can refer. By the middle of the first semester students, individually and in teams, have begun accumulating their own "war stories"; some positive, some negative. This personalized knowledge provides a solid base for more advanced concepts.

Acknowledgements

We would like to acknowledge Dr. Suzanne Smith's contribution in the development of the research grant proposal used to support this work. East Tennessee State University provided institutional support throughout the project. A special debt is owned to the software engineering students who survived early versions of this course and helped us develop a better product. We would like to thank the Defense Advanced Research Project Office and the Ada Joint Program Office for their support of the improvement of software engineering curricula and the support by the U.S. Army Research Office.

References

[Adams 93] E. Adams, "Experiences in Teaching a Project-Intensive Software Design Course," Proceedings of the First Annual Rocky Mountain Small College Computing Conference, volume 8, number 4, March 1993, pp. 112-121.

[ANSI/MIL-STD-1815a, 1983] ANSI, American National Standard reference manual for the Ada programming language, ANSI, New York, New York, 1983.

[AST 90] School of Applied Science and Technology Language Skills Handbook, East Tennessee State University, 1990.

[Benjamin 1991] G. Benjamin, Ada Minimanual, to Accompany Appleby:Programming Languages, McGraw-Hill, Inc,New York, N.Y., 1991

[Booch 93] G. Booch, Software Engineering with Ada, Benjamin/Cummings Publishers, Menlo Park, CA, Forthcoming.

[Brooks 82] F. Brooks, The Mythical Man Month, Addison-Wesley, Reading, MA, 1982.

[Bruegge 91] B. Bruegge, J. Cheng, and M. Shaw, "A Software Engineering Project Course with a Real Client," CMU/SEI-91-EM-4.

[Callis 91] F.W. Callis and D.L. Trantina, "A Controlled Software Maintenance Project," Software Engineering Education, SEI Conference 1991, Pittsburgh, PA, October 7-8, 1991, Springer-Verlag, New York, NY, pp. 25-32.

[Deimel 90] L.E. Deimel and J.F. Neveda, "Reading Computer Programs: Instructor's Guide and Exercises," CMU/SEI-90-EM-3.

[Deimel 91] L.E. Deimel, "Scenes from Software Inspections," CMU/SEI-91-5.

[Engle 89] C.B.Engle, G. Ford, and T. Korson, "Software Maintenance Exercises for a Software Engineering Project Course," CMU/SEI-89-EM-1.

[Ford 87] G. Ford, N. Gibbs, and J. Tomayko, "Software Engineering Education: An Interim Report from the Software Engineering Institute," SEI-87-TR-8.

[Herro 88] John Herro, The Interactive Ada-Tutor, Software Innovations Technology, 1083 Mandarin Drive N.E., Palm Bay FL. 32905-4706

[Humphrey 88] W. S. Humphrey, "Characterizing the Software Process: A Maturity Framework," IEEE Software, March 1988, pp. 73-79.

[NASA 86] NASA Sfw-DID-41, "Lessons Learned Document Data Item Description."

[Shaw 91] M. Shaw and J. Tomayko, "Models for Undergraduate Project Courses in Software Engineering," Software Engineering Education, SEI Conference 1991, Pittsburgh, PA, October 7-8, 1991, Springer-Verlag, New York, NY, pp. 25-32.

[SPC 91] Software Productivity Consortium, Ada Quality and Style: Guideline for Professional Programmers, Software Productivity Consortium, Herndon, Virginia, 1991.

[Stoecklin 93] S. Stoecklin, Ada Laboratory Exercises, funded by a Darpa Grant 1993.

[Stuckenbruck 81] L.C. Stuckenbruck, A Decade of Project Management, Project Management Institute, 1981.

[Tomayko 91] J.E.Tomayko, "Teaching Software Development in a Studio Environment," SIGCSE Bulletin, volume 23, number 1, March 1991, pp. 300-303.

e. Integrated Course Structure

Three elements need to be carefully integrated to make this course a success. Lectures to the students about the principles and concepts of software engineering must be coordinated with their project tasks and the labs must illustrate both lecture materials and clarify project tasks. Our model for this integration is illustrated in the matrix below. The Roman numerals in the first column indicate the semester, the numbers in the lecture column correspond to the lecture numbers in section II of this document, the lab numbers correspond to the lab numbers in section III of this document, and the configuration items for the various student items correspond to the configuration items for their projects. These configuration items are described in section IV d.

Week	Lecture	Lab	Project
I-1	001 - Intro, syllabus, policies,overview		
I-1	002 - Definitions of SE, software life cycle, quality, process		
I-2	003 - Requirements extraction - what & how	001 - Small project customer request, team organization, project mgmt plan	Begin CI-1
I-2	004 - Intro to structured analysis, context diagram, DFDs, data dictionary	002 - CD, DFD exercise	CI-1
I-3	005 - Quality standards in requirements,requirements extraction, DFDs	003 - Feedback on CI-1, small project CD, DFD, DD	Begin CI-2
I-3	006 - Intro to design	04A - Structure charts	CI-2
I-4	007 - DFDs and DD, structure charts, test plans and requirements traceability	005 - CI-3 for small project, Feedback on CI-2	Begin CI-3
I-4	008 - Design concepts; architectural, behavioral, procedural design	006 - Additional Feedback on CI-2, preparation for design review	CI-3
I-5	009 - Testing and test plans	007 - Development of classes of tests for small project	
I-5	010 - Ada and design; Ada as a design notation	008 - Design review presentation	CI-4

Week	Lecture	Lab	Project
I-6	011 - Software maintenance	009 - Feedback on design review presentation	
I-6	012 - Controlling disciplines; configuration management	010 - Feedback on CI-5, preparation for acceptance test	
I-7	013 - Ada and maintenance	011 - Extended project customer request	CI-5
I-7	014 - Software life cycle models	012 - Extended project team organization, ,	CI-6 CI-7
I-8	015 - Requirements elicitation, analysis and specification	013 - Peer review and acceptance test /review	CI-8
I-8	016 - Ada as a specification and maintenance tool	014 - Instructors assessment of small project,team asssignments, distribute SPMP	
I-9	017 - Requirements standards, 2167A	015 - User project perspective	
I-9	018 - Team organization and software quality	016 - Tasks for configuration manager, requirements, user interface, and test plan teams	
I-10	019 - Examination I-1	(Examination I-1, cont)	
I-10	020 - Return/discuss first examination; from ERD's to Ada	017 - Presentation/review of configuration management plan	CI-1
I-11	021 - Verification and validation	018 - Preliminary requirements review; Preliminary user manual, user interface review	
I-11	022 - Testing	019 - Preliminary test plan review	
I-12	023 - More on structured analysis; process specifications	020 - Requirements review	CI-2 CI-3 CI-4
I-12	024 - Transform analysis, transaction analysis		CI-5

Week	Lecture	Lab	Project
I-13	025 - Coupling and cohesion		
I-13	026 - Preliminary design using functional decomposition; intro to object-orientation, object - oriented analysis	021 - Steer preliminary design	
I-14	027 - High level OOD, identifying objects, Rumbaugh notation	022 - Ada laboratory environment	
I-14	028 - Ada packages	023 - Peer review of extended project through preliminary design; preliminary design review	
I-15	029 - Software quality assurance and reviews	024 - User manual/interface and test plan reviews	
I-15	030 - Review standards, review checklists		CI-6 CI-7 CI-8
I-16	Final Examination I-2		
II-1	031 - High level design vs detailed design; Detailed design deliverable and procedures	025 - Review extended project specifications and preliminary design completed in semester 1;	
II-1	032 - Reuse	026 - reorganize project and teams	
II-2	033 - Nassi-Shneiderman diagrams	027 - Nassi-Shneiderman diagrams; project teams work	
II-2	034 - Ada: text I/O		
II-3	035 - Ada: data types		
II-3	036 - Ada: statements, control structures 037 - Ada: structured data types		
II-4	038 - Ada: access data types	028 - Detailed design review	
II-4	039-Ada procedures, functions, packages. 040- Ada Generics	029 - Feedback on detailed design	

Week	Lecture	Lab	Project
II-5	041 - Ada: exceptions	030 - Videotape on code inspections	
II-5	042 - Ada: sequential and direct files		
II-6	043 - Ada: Tasks		
II-6	* see note		
II-7	044-Examination II-1		
II-7	045-Review Examination II-1	031 - Maintenance project description, team organization, team assignments, assignment of maintenance 1	
II-8	* see note		
II-8	* see note		
II-9	046 - Use cases	032 - Code inspections	
II-9	*see note	033 - Maintenance project: review task 1, assignment of task 2	
II-10	*see note		
II-10	* see note		
II-11	* see note	034 - Maintenance project: review task 2, assignment of task 3	CI - 10
II-12	* see note	035 - Extended project: system acceptance test	CI -11
II-12	* see note	036 - Feedback, instructors assessment of acceptance test and extended project	
II-13	047 - Implementation languages		
II-14	048 - Project scheduling, work breakdown structures		
II-14	049 - Project estimation, COCOMO	037 - Estimation: function points wrt extended projects	

Week	Lecture	Lab	Project
II-15	* see note	038 - Individual and small group analysis of ethical scenarios	
II-15	050 - Course assessment		
II-16	Final Examination II		

* NOTE Meetings during these weeks are used to meet the needs of the extended project. They can be utilized for the various reviews, individual team meetings, or project meetings with the entire class.

e. Acknowledgements

The Defense Advanced Research Project Agency and the Ada Joint Project Office provided the opportunity to undertake this project. We appreciate their interest in and commitment to undergraduate software engineering education and their continued support throughout the project. Likewise we appreciate the support of the U.S. Army Research Office.

This course was developed and tested over several semesters at East Tennessee State University. The students in these classes played vital roles in helping us develop a better product. In particular a special debt is owed to the following students for their intensive efforts during the summer of 1993: Woody Beverly, Darlene Fladager, Mitch Moses, Drew Picklesimer, Kellie Price, Eugene Price, James Stephenson, and Mike Stillwell.

Bob Tolbert provided technical assistance and brought consistency to the documentation of the lectures, laboratories, and supporting materials. Luke Pargiter's expertise and patience in providing hardware and software support was invaluable.

Finally, we appreciate the institutional support provided by East Tennessee State University in the form of released time, laboratory facilities and support, administrative support, and commitment to undergraduate software engineering curriculum.

Real-World Software Engineering

II LECTURES

a. Lecture format and lecture forms.

The course, covering two semesters, consisted of formal lectures, discussions, laboratory meetings during scheduled class meetings and team meetings outside of scheduled class hours. This section contains the lectures in the order in which they are given to the class. The two semesters are divided up into two class meetings per week. The amount of time given to formal lecture during these meetings varied depending upon the project stage and students understanding of and progress on project deliverables. Because of difficulties in establishing team meeting times, some class time was devoted to team meetings. Using class meeting time in this way also offers the teacher an opportunity to participate in the meetings. Because the projects are the major scheduling factor in this course, it is important to be flexible in terms of trying to cover two lectures every week. Project reviews, e.g., requirements reviews, design reviews, and test plan reviews, are most effective when the entire class participates. These reviews consume class meeting time. We have found that this course must rely on the students doing the required readings. Because of the other events which use class time, the student can not depend on the teacher to cover every concept during formal lecture.

The lecture forms provide most of the structure for a lecture and significant detail for each lecture. The form starts out with the general topics for the lecture. These are stated in terms of the concepts that are addressed in the lecture. They are followed by the instructional objectives of that particular lecture. The objectives are generally stated in terms of behavioral goals. Both the topics and the objectives can be used in test construction. The topics can be used to construct concept questions and the objectives can be used to construct performance questions.

We have used the SET UP, WARM-UP section to provide some connection between the current lecture and a previous lecture or topic. In some cases, e.g. when there are several lectures on Ada syntax, we have not provided such a connection.

The CONTENTS section contains the main body of the lecture. The topics are presented in several paragraphs. The overheads used for that lecture follow the CONTENTS section. As a topic is described in the CONTENTS section, the related overhead is named by using both the lecture number and an overhead number, e.g.,L23OH2. This refers to the second overhead used in lecture 23. The overheads are formatted for easy duplication. Overheads generally contain examples of the concept being discussed in the CONTENTS section. They can also contain sample exercises to be done during class in order to reinforce concepts just discussed. The CONTENTS section contains answers to the exercises on the overheads. In a few cases, such as the sample test plan overhead, we have included rather lengthy explanations of the overhead in the form of instructor notes.

The PROCEDURE section contains subsections on teaching method and vocabulary introduced. The presumption is that the major teaching method is lecture and discussion using the overheads and handouts included in the lecture forms. In several cases we have included some hints at additional discussion or alternative teaching methods that we have had success with when covering a particular topic. The vocabulary introduced can be provided to the student as a review tool.

The RELATED LEARNING ACTIVITIES section list the particular labs which we have associated with a particular lecture. These labs frequently tie the theoretical lecture material to the practical concerns of their projects.

There are two sections on readings, one is the assigned reading in the textbooks we use for the course, and the other is a list of reading on the same topic in other software engineering textbooks. If we cited the work of another software engineer in the lecture or overhead material, then a reference to that material is included in the reading list.

On some occasions, the contents refer to an overhead used in a previous lecture, e.g., lecture five refers to an overhead from lecture 3. When this happens, the earlier overhead is also included in the current lecture. Lecture 5 has an overhead from lecture 3 in it.

These forms should provide you with an adequate foundation for structuring you lectures and relating them to the significant experiential elements of this course.

b. Lecture Forms

LECTURE NUMBER: 001

TOPIC(S) FOR LECTURE:
Introduction to course

INSTRUCTIONAL OBJECTIVE(S):

1. Learn names of instructor(s), and other students.

2. Learn course format, and course policies.

3. Become familiar with detailed course syllabus.

SET UP, WARM-UP:
(How to involve the learner: recall, review, relate)

Try to set tone for the rest of the course. In previous courses you have covered many aspects of software development (mention specific topics such as programming, design; mention specific courses). We'll be looking at those as well as others as we consider software as an engineered product; some topics will be completely new, some technical, some non-technical.

Realistic team project experiences will be integrated into the course. Since the instructor(s) and students will be spending a lot of time together in the classroom and in your team projects, it is important to get to know one another and to develop a comfortable working atmosphere.

(Learning Label- Today we are going to learn ...)

Today we want to give a sense of the course; specifically an overview including how class and project activities will be integrated, the course format, syllabus, and course policies.

CONTENTS:

1. Introductions
    a. Professor(s) introduce themselves and others responsible for the course.

    b. Have the students introduce themselves since they will be working together on projects.

2. Approach

   a. This is a two-semester undergraduate software engineering class that presents a thorough coverage of software engineering while at the same time providing meaningful project experiences that mimics a real-world software engineering process. You will each get to work with several people in a vaariety of roles on several projects.

   b. We are taking a "spiral approach" to the material presented. Our first pass through some topics will be exactly that - a first pass. We intend the depth provided to be sufficient for application to your first project. Gaps will be filled in during subsequent passes in the spiral. Similarly there are many techniques and methodologies for analysis and design but we need to choose specific ones to apply to the first project in a timely fashion. So, don't worry that we're moving on to design and you feel that there are many aspects of analysis that have not been covered.

   c. The understanding that project activities and class activities will be carefully coordinated is important. It is also important to make students aware that factors in the success or failure of software projects include non-technical problems as well as technical problems.

3. Distribute and discuss course policies.
   L1HD1
   a. Course description and typical class format.

   b. Prerequisites.

   c. Textbooks.

   d. Grading policies.

4. Questions from students?

5. Distribute and discuss detailed course syllabus.
   L1HD2
   a. Walk through first week of the sylabus in order to explain all aspects and notation.

6. Questions from students?

PROCEDURE:

teaching method and media:

Set tone for course; try to generate enthusiasm, team concept, opportunities to gain realistic project experience.

Read D. Gotterbarn and Robert Riser, "Real-World Software Engineering: A Spiral Approach to a Project-Oriented Course," Lecture Notes in Computer Science 750, Software Engineering Education, ed Jorge L. Diaz-Herrera, Springer Verlag 1994 pp 119-151 for a complete understanding of the structure of this course.

vocabulary introduced:

INSTRUCTIONAL MATERIALS:

overheads:

handouts:

L1HD1	Course policies
L1HD2	Detailed course syllabus

other:

RELATED LEARNING ACTIVITIES:

READING ASSIGNMENTS:

RELATED READINGS:

COURSE POLICIES

COURSE DESCRIPTION: This is a two-semester software engineering course that covers all aspects of the software development process while providing participants with realistic project experiences. Each student will function on multiple project teams in a variety of roles and responsibilities.

Project and lecture activities will be coordinated. Typical class meetings will consist of a lecture component and a lab component. The lab component will include both individual activities and project team activities.

PREREQUISITES: File Processing, Data Structures

TEXT:
a) Software Engineering 4th edition, Sommerville
b) Software Engineering with Student Project Guidance, Mynatt
c) Ada Minimanual, Benjamin

GRADING:

Tests (2, equally weighted)	40%
Team project 1	15%
Team project 2	25%
Participation:	20%

Includes attendance, class discussion,exercises and assignments, quizzes on assigned reading, presentation responses

A passing average on each of the four components above is required to pass the course.

Individual project grades will be based on 3 factors: team project grade, peer review, and instructors' perceptions of individual contributions.

All deliverables are due at the start of class on the specified due date unless otherwise stated.

GRADING SCALE:

93 -100: A	77 - 79: C+	0 - 54: F
90 - 92: A-	70 - 76: C	
	65 - 69: C-	
88 - 89: B+		
83 - 87: B	60 - 64: D+	
80 - 82: B-	55 - 59: D	

DETAILED COURSE SYLLABUS

Week One

Topics:

Introduction to course, course objectives, workshop format, grading policies, and team projects

Introduction to software engineering, quality software, requirements from the viewpoints of the customer and user, development of abstract and requirements list from problem specification, and the example project

Readings for class:

Sommerville	Chapter 1 (pp. 1-5)
Mynatt	Chapter 1 (pp. 1-27)

Week Two

Topics:

requirements extraction

analysis process

context diagrams, data flow diagrams (DFDs), and data dictionary

Readings for class:

Mynatt	Chapter 2 (pp. 44-62 and pp. 70-74)
Sommerville	Chapter 3 (pp. 47-63)

Week Three

Topics:
- quality standards in requirements
- requirements extraction
- function-oriented design - more on DFDs and data dictionaries, structure charts

Readings for class:

Sommerville	Chapter 3 (pp. 47 -63)
Sommerville	Chapter 10 (pp. 171-188)
Sommerville	Chapter 12 (pp. 219-237)
Mynatt	Chapter 2 (pp. 44-62)
Mynatt	Chapter 4 (pp. 143-156)

Week Four

Topics:
- data flow diagrams and data dictionaries
- structure charts
- requirements traceability

Readings for class:

Sommerville	Chapter 12 (pp. 219-234)
Sommerville	Chapter 10 (pp. 171-188)
Mynatt	Chapter 4 (pp. 143-156)

Week Five

Topics:
- testing and test plans
- Ada and design notation

Readings for class:

Sommerville	Chapter 19 and 22 (pp.378-388 and pp.425-441)
Mynatt	Chapter 7 (pp.276-315)
Sommerville	Appendix A (pp.607-620)

Week Six

Topics:
- software maintenance
- configuration management and software quality assurance (SQA)

Readings for class:

Sommerville	Chapter 28 (pp. 533-541)
Sommerville	Chapter 29 (pp. 551-564)
Mynatt	Chapter 8 (pp. 334-340)

Week Seven

Topics:
- Ada and maintenance
- software life cycle models

Readings for class:

Sommerville	Chapter 1 (pp. 5-18)
Mynatt	Chapter 1 (pp. 12-27)

Week Eight

Topics:

requirements analysis and specification - client requests, definition of requirements, requirements specification

Ada as a specification tool and a maintenance tool

Readings for class:

Sommerville	Chapter 5 (pp. 85-103)
Mynatt	Chapter 2 (pp. 62-83)

Week Nine

Topics:

requirements standards, 2167a

team organization and software quality

Readings for class:

Sommerville	Chapter 3 (pp. 45-61)
Sommerville	Chapter 5 (pp. 85-103)
Mynatt	Chapter 2 (pp. 62-91)
Mynatt	Chapter 1 (pp. 31-42)

Week Ten

Topics:

Examination I-1

ERDs and Ada

Readings for class:

None

Week Eleven

Topics:
- verification and validation (V&V)
- testing

Readings for class:

Sommerville	Chapter 19 (pp. 373-386)
Sommerville	Chapter 22 (pp. 425-439)
Sommerville	Chapter 23 (pp. 441-454)
Sommerville	Chapter 24 (pp. 457-473)
Mynatt	Chapter 7 (pp. 274-316)

Week Twelve

Topics:

relationship between requirements and preliminary design, more on structure charts, transform analysis, transaction analysis, designing data structures, abstraction

Readings for class:

Sommerville	Chapter 2 (pp. 71-82)
Sommerville	Chapter 12 (pp. 222-228)
Mynatt	Chapter 4 (pp. 62-69)
Mynatt	Chapter 4 (pp. 143-169)

Week Thirteen

Topics:
- introduction to object-oriented development
- coupling and cohesion

Readings for class:

Sommerville	Chapter 10 (pp. 182-188)
Mynatt	Chapter 3 (pp. 94-130)
Mynatt	Chapter 4 (pp. 144-150)

Week Fourteen

Topics:
- high-level object-oriented design
- notation for preliminary design
- Ada packages

Readings for class:

Benjamin	Chapter 8 (pp. 73-78)
Sommerville	Chapter 10 (pp. 177-182)
Sommerville	Chapter 11 (pp. 194-236)
Sommerville	Appendix A (pp. 610 -613)
Mynatt	Chapter 8 (pp 364-368)

Week Fifteen

Topics:
- Introduction to software quality assurance
- Reviews - walkthroughs and inspections
- Review standards and checklists

Readings for class:

Sommerville	Chapter 31 (pp. 589-598)
Mynatt	Chapter 2 (pp. 77-79)

Week Sixteen

FINAL EXAMINATION I

Week Seventeen

Topics:
- reliability and reuse in detailed design
- The relation between detailed and high-level design
- detailed design procedures
- detailed design deliverables

Readings for class:

Sommerville	Chapter 16 (pp. 309-328)
Mynatt	Chapter 1 (pp. 77-79)
Mynatt	Chapter 3 (pp. 94-138)
Mynatt	Chapter 4 (pp. 169-183)
Benjamin	Chapters 9 and 12 (pp. 79-85 and 111-117)

L1HD2

Week Eighteen

Topics:
- Nassi-Shneiderman chart notation
- Introduction to Ada
- I/O in Ada

Readings for class:

Mynatt	Chapter 5 (pp. 198-202)
Benjamin	Chapter 1 (pp. 1-10)

Week Nineteen

Topics:
- Ada data types
- Ada statements
- Ada structured data types

Readings for class:

Benjamin	Chapters 2-3 (pp. 11-28)
Benjamin	Chapter 4 (pp. 29-37)
Benjamin	Chapter 5 (pp. 39-50)

Week Twenty

Topics:
- access data types in Ada
- Ada procedures, functions, and packages
- Ada generics

Readings for class:

Benjamin	Chapter 7 (pp. 63-72)
Benjamin	Chapters 6 and 8 (pp. 51-62 and 73-78)
Benjamin	Chapter 9 (pp. 79-87)

Week Twenty-one

Topics:
- exceptions and exception handlers in Ada
- sequential and direct files in Ada

Readings for class:

Benjamin	Chapter 10 (pp. 89-96)
Benjamin	Chapter 12 (pp. 111-117)

Week Twenty-two

Topics:
- tasks in Ada
- Project meetings * see note

Readings for class:

Benjamin	Chapter 11 (pp. 97-109)

Week Twenty-three

Topics:
- Examination II-1
- Handback and review examination II-1

Week Twenty-four

Topics:
- project meetings * see note

Readings for class:
- none

Week Twenty-five

Topics:
- introduction to use cases
- project meetings * see note

Readings for class:
- none

Week Twenty-six

Topics:
project meetings * see note

Readings for class:
none

Week Twenty-seven

Topics:
project meetings * see note

Readings for class:
none

Week Twenty-eight

Topics:
implementation languages - project driven choices

Readings for next class:

Mynatt	Chapter 5 (pp. 207-235)
Mynatt	Chapter 6 (pp. 239-271)

Week Twenty-nine

Topics:
project scheduling
work breakdown structures
software project management (SPM) - planning, scheduling
COCOMO
code estimation techniques

Readings for class:

Sommerville	Chapter 25 (pp. 477-492)
Sommerville	Chapter 26 (pp. 495-507)
Sommerville	Chapter 27 (pp. 511-533)
Mynatt	Chapter 1 (pp. 17-27)

Week Thirty

Topics:
professionalism, ethical issues
course assessment

Readings for class:
Sommerville Chapter 21 (pp.407-425)

Week Thirty-one

FINAL EXAMINATION II

*** NOTE**

Meetings during these weeks are used to meet the needs of the extended project. They can be utilized for the various reviews, individual team meetings, or project meetings with the entire class.

LECTURE NUMBER: 002

TOPIC(S) FOR LECTURE:

Introduction to software engineering, quality software, life cycles, and process models.

INSTRUCTIONAL OBJECTIVE(S):

1. Understand software crisis, software engineering, quality software, and process models.
2. Realize the reasons leading to a software crisis and the emergence of software engineering.
3. Understand the attributes of quality software.
4. Recognize the different viewpoints in the development of software.

SET UP, WARM-UP:
(How to involve the learner: recall, review, relate)

The importance of developing quality software is related in this lecture to the projects. The students begin thinking about the importance and role of maintenance in the development of large software systems. The concept of having a good process by which to develop the products is introduced.

(Learning Label- Today we are going to learn ...)

The introduction of software engineering is related to the students' previous experience with developing software in other classes. The idea of programming in the small (previous experience) versus programming in the large (software engineering) is expressed. Emphasis is placed on the fact that software development in a real-life situation is a team effort.

CONTENTS:

1. Software Crisis

L2OH1

a. The phrase "software crisis" was coined in the late 1960's at a conference which was addressing the problems of software development. It refers to a series of problems with software development practices including: the inability to deliver software within budget, on schedule, and meeting customer needs.

L2OH2

b. Factors contributing to the software crisis are described.

2. Software Engineering

L2OH3

a. In defining "software engineering" one must consider what is meant by software (the products -- the source code and the internal and external documentation needed for development, installation, utilization, and maintenance) and what is meant by engineering (the process -- the application of a systematic and measurable approach).

L2OH4

b. Software engineering is needed for the development of large, complex software systems that are developed by teams rather than individuals, that require understanding of the technical and nontechnical aspects of software development, and that require project management and effective user interface.

3. Quality Software

L2OH5

a. The primary goal of software engineering is the production of quality software (i.e., well-engineered software).

L2OH6

b. The attributes of quality software are not an agreed upon list of characteristics. The attributes are often dependent on the point of view of the person involved (e.g., sponsor/customer, user, maintainer). It is the developers job to satisfy these multiple perspectives.

4. Software Development Life Cycle

L2OH7

a. The activities required throughout the life cycle of a software system are divided into stages with each stage having its own set of activities. The manner, in which these stages are organized, is the process model. Different organizations of these stages lead to different process models.

L2OH8
L2OH9

b. The stages of the waterfall model. Use this opportunity to compare and contrast the different stages of software development.

L2OH10

c. The stages of the prototype model

L2OH11

5. Difficulties in Software Development

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

It is important to touch on all the stages of the life cycle here to give a general overview of software developement. The details of these stages will be presented in later lectures.

<u>vocabulary introduced</u>:

software crisis
software engineering
quality software
sponsor/customer
user
maintainer/modifier
process models
requirements
waterfall model
analysis
design
testing
maintenance
prototyping

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>:

L2OH1	Software crisis
L2OH2	Factors that contribute to the software crisis
L2OH3	Definitions of Software engineering
L2OH4	The concerns of software engineering
L2OH5	Characteristics of quality software (Sommerville)
L2OH6	Perspectives on Software Quality (Mynatt)
L2OH7	Software development life cycle
L2OH8	Development Models
L2OH9	Waterfall model
L2OH10	Prototyping model
L2OH11	Difficulties in software development

<u>READING ASSIGNMENTS</u>:

Sommerville Chapter 1 (pp. 1-5)
Mynatt Chapter 1 (pp. 1-27)

RELATED READINGS:

Berzins Chapter 1 (pp. 1-3)
Booch Chapter 4 (pp. 27-31)
Booch(2) Chapter 2 (pp. 17 -20)
Ghezzi Chapter 2 (pp. 17-40)
Pressman Chapter 1 (pp. 3-36)
Schach Chapter 3 (pp. 47-70)

Software Crisis

Problems encountered in the development of <u>large</u> software systems

Over Budget

Behind Schedule

Failure To Meet Customer Needs

Low Quality

L2OH1

Factors Contributing To Software Crisis

Inability to predict time, effort, and cost in software development

Poor quality of software

Changes in the ratio of hardware to software cost

Increasingly important role of maintenance

Advances in hardware and software

Demand for larger and more complex software

Definitions of Software Engineering

IEEE: the systematic approach to the development, operation, maintenance, and retirement of software

Pressman: the establishment and use of sound engineering principles in order to obtain, economically, software that is reliable and works efficiently on real machines.

Fairley: the technological and managerial discipline concerned with the systematic production and maintenance of software products that are developed and modified on time and within cost estimates. The primary goals are to improve software quality and to increase productivity.

Gotterbarn/Riser/Smith: the planning, development, and maintenance of computerized solutions to real problems. It encompasses techniques which treat software as an engineered product requiring planning, analysis, design, construction, testing, documentation, maintenance, and management.

Software engineering concerned with

Technological and managerial aspects of software development

Systematic production and maintenance of software

Developing software on time and within budget

Assuring software quality

Assuring software reliability

Reducing costs

Increasing productivity

Increasing benefits

Software as an engineered product; attention to process as well as product

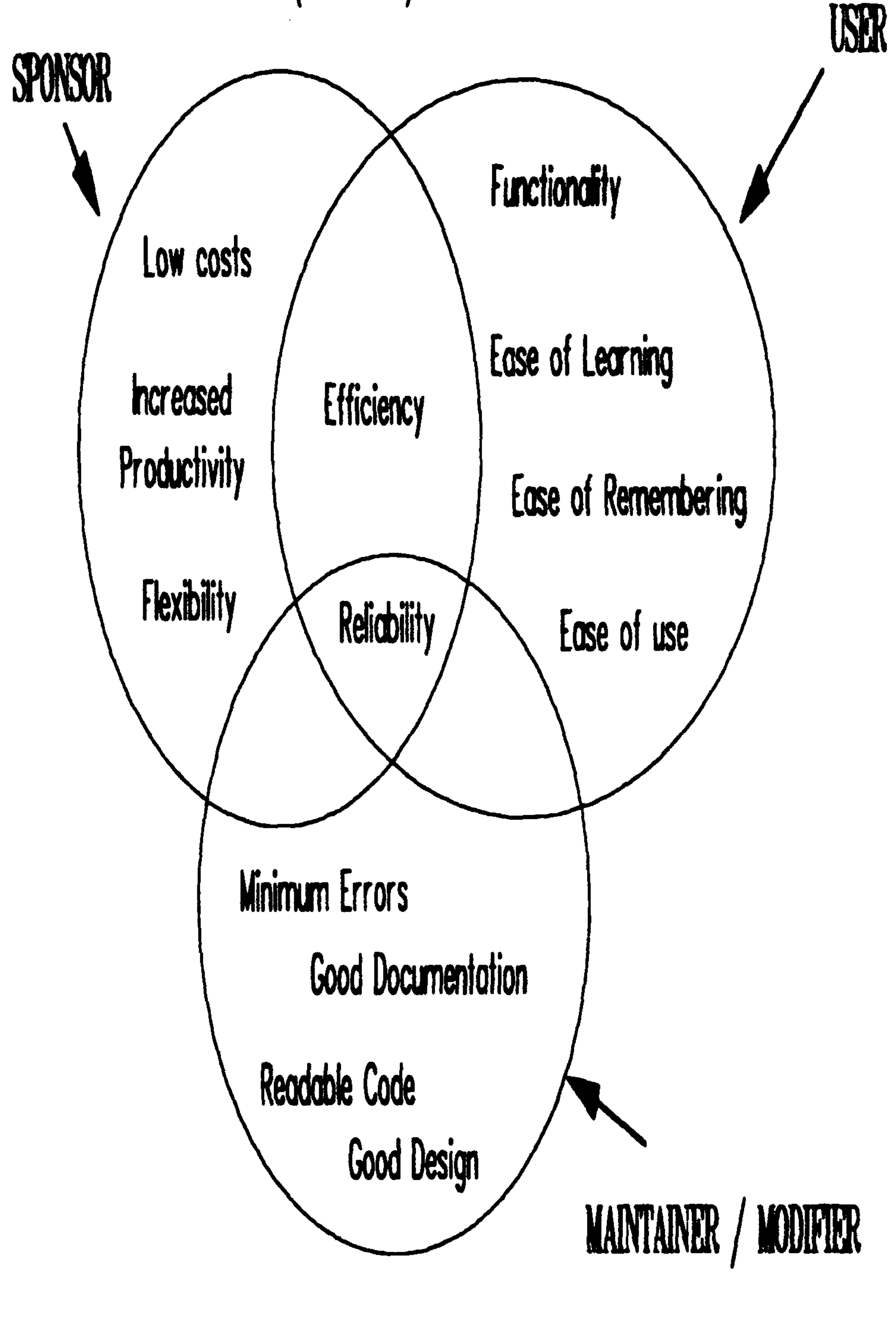
Quality Software

Well-engineered

Attributes:

1. provides required functionality
2. should be maintainable
3. should be reliable
4. should be efficient
5. should offer appropriate user interface
6. should be cost effective

MORE CHARACTERISTICS OF QUALITY SOFTWARE (MYNATT)



Software Development Life Cycle

The activities involved in the production of a software system

The development, operation, maintenance, and retirement of software

Development activities include:

- requirements analysis and specifications
- design
- implementation
- system testing
- installation
- maintenance

Development Models

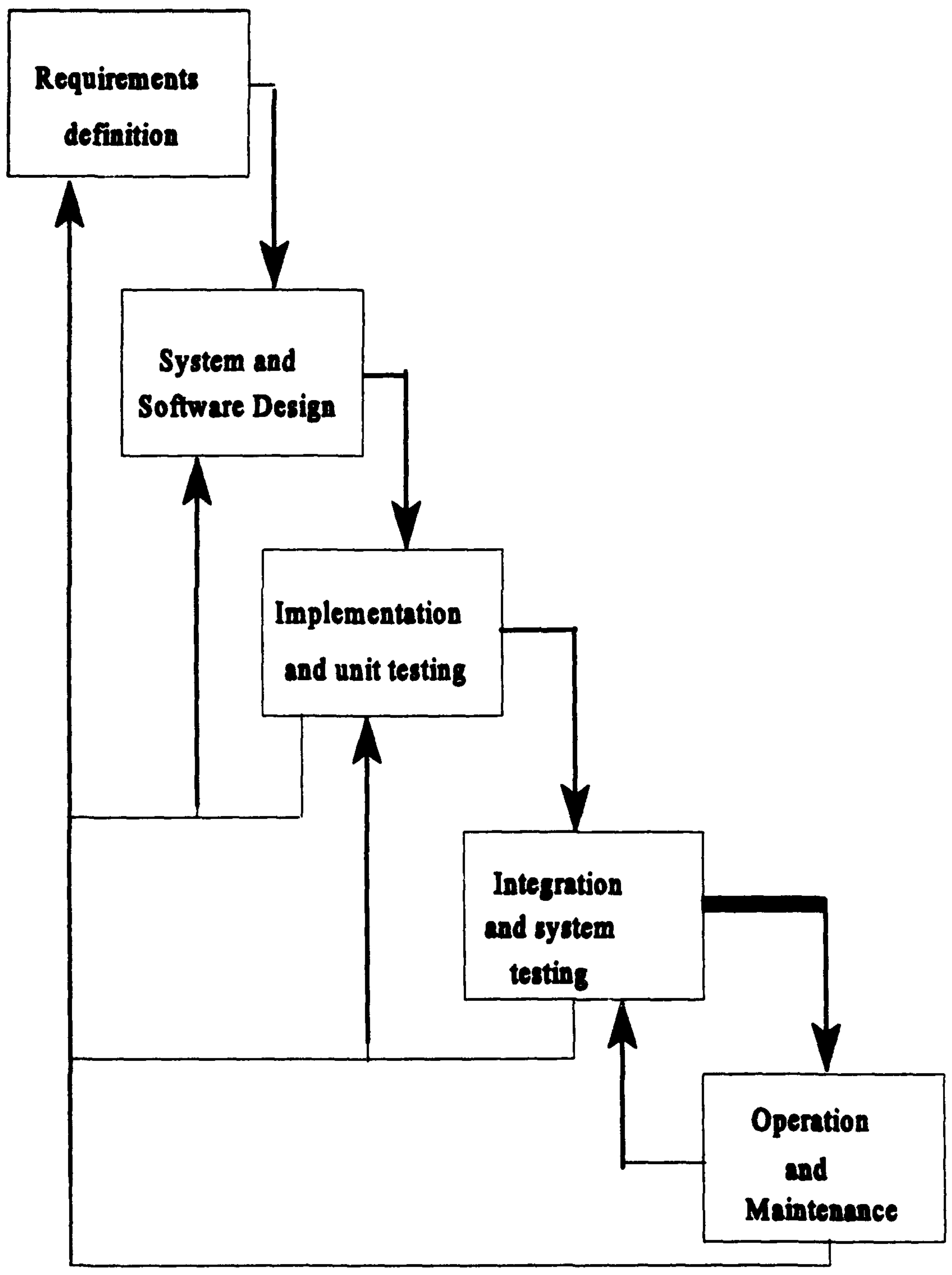
Ways in which the set of steps in software engineering are applied

Examples of software process models:

waterfall model

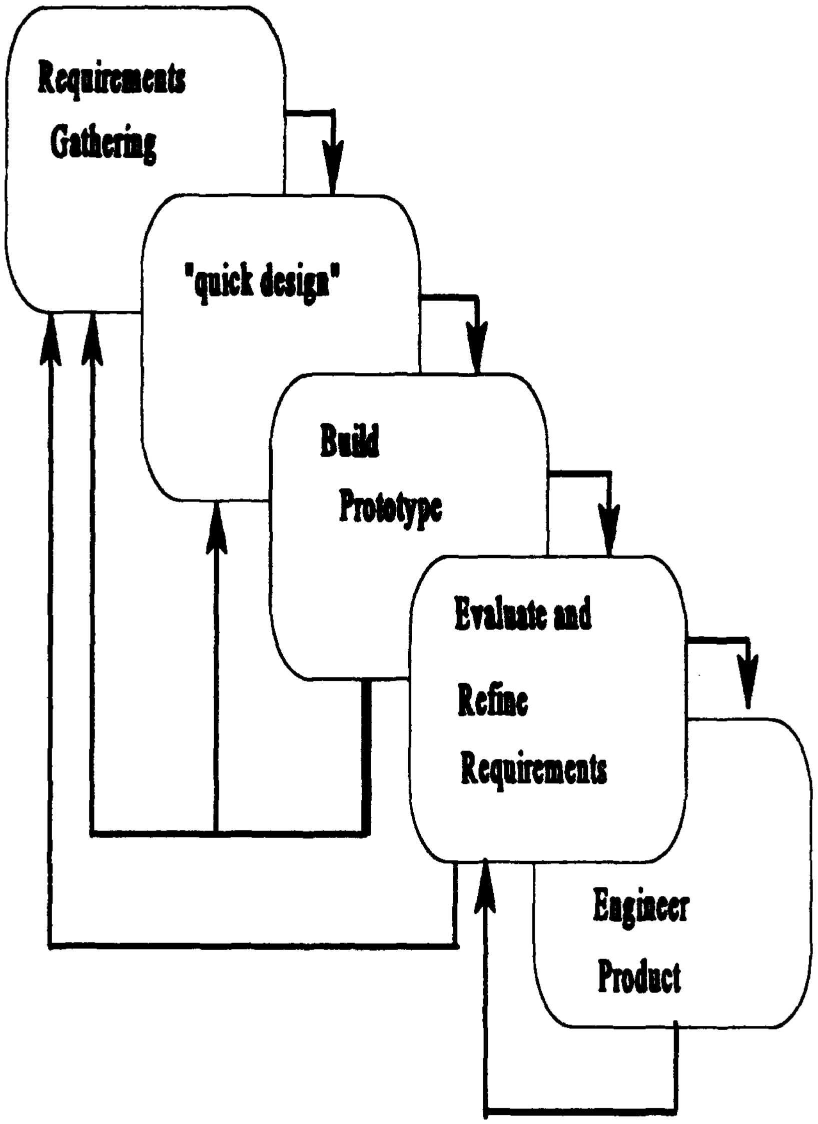
prototyping

Waterfall Model


Requirements definition
System and Software Design
Implementation and unit testing
Integration and system testing
Operation and Maintenance

L2OH9

Prototyping



Difficulties in Software Development

Communications

Sequential nature of system development

Project characteristics

Characteristics of personnel

Management issues

LECTURE NUMBER: 003

TOPIC(S) FOR LECTURE:

Requirements Analysis and Specification
The importance and the difficulty of requirements extraction
A method for doing requirements extraction

INSTRUCTIONAL OBJECTIVE(S):

1. Understand the difficulty of gathering and specifying requirements
2. Do a preliminary requirements abstraction
3. Develop a requirements list

SET UP, WARM-UP:

(How involve learner: recall, review, relate)

In your other courses, such as data structures, you were given detailed descriptions of the tasks your program was supposed to perform. Sometimes the overall structure of the program was also provided for you. Given such clear descriptions, you were optimistic about your ability to quickly write programs which fit the guidelines of those program descriptions. This is not the way most programming tasks really start. Suppose one of your friends asked you to help her develop a program which she was writing for a friend. What would be the first thing you would ask about the program? (List several of the responses on the board. What you are looking for is -- What is the program supposed to do? or What are its functions?) The answer to this question about functions is often given in what is called a problem specification of a client's request.

(Learning Label- Today we are going to learn ...)

Today we are going to look at such a request as it might be made by someone who wants a computer system developed for them. We shall see how difficult it is to arrive at those problem descriptions. These problem descriptions are the basis for the program descriptions which you take for granted. From this problem description we will start to develop a list of the desired functions. This list is sometimes called a systems requirement list.

CONTENTS:

1. Introduce the concept of a problem specification

    a. Give class examples of preliminary problem specifications, e.g., "Build a house for me which will hold my children", "Write a program which will control the temperature in an incubator for premature children", "Build a dream house for your parents -- money is no object", "Describe a centralized college registration system."

2. Discuss the problems with such specifications and discuss why each difficulty occurs.

   a. Vagueness , insufficient detail - the client assumes a familiarity with the problem domain which you don't have.

   b. Ambiguity - can be caused by 2a, or client is not aware of other possibilities, or the problem can be described from multiple perspective. People are not used to communicating at the required level of precision.

   c. Incomplete - functions are missing because they were not thought of and combinations of conditions were not considered.

   d. Instability of description -problem descriptions vary over time because of changing conditions.

   e. Discuss the role of a professional software engineer in attempting to resolve these difficulties.

3. The goal of requirements extraction is to solve these problems by completely defining the problem space--WHAT is required. A first step is to separate out all of the functionality in the original request.

   a. Extracting all of the functions in the client request helps one identify the problems in the original request. The functions can be recorded in a system requirements list which later become part of a Software Requirements Specification (SRS).

   b. This results in a complete description of the <u>external behavior</u> of the product. This initial list of the external behavior of the system needs to be refined by removing the difficulties of 2a-2e. This refinement require communication with and participation of the client.

4. Hints for developing a requirements list.

   a. In order to better understand the product, try to visualize it in action.

   b. List "what a product does" rather than "how it does it."

   c. Understand the domain in which the product operates.

d. Look for functional requests by analyzing the client request for verbs.

5. Distribute the Preliminary Client Request for the KoFF System.

L3HD1

a. Read through it with the class.

b. Ask them how they would go about building the system and direct the discussion toward building a requirements list.

c. Begin to identify functions by working through the first two paragraphs looking for verbs and other indications of what the system does. Write some of the requirements for these identified functions.

L3OH1

d. Show them the preliminary requirements list for KoFF and discuss its imperative structure.

e. Discuss the adequacy of this list. Sample problems are contained on the instructor's copy of the preliminary requirements list. In discussing the requirements, lead the students toward the need for testable requirements. (One technique is to divide the class into groups and have each group review the adequacy of this list and report their findings to the class as a whole.)

6. Discuss multiple viewpoints of this system.

a. The customer is satisfied with many aspects of the system, e.g., the customer is pleased that the user is charged for a tape even before it is dispensed.

b. The user is unsatisfied with many aspects of the system, e.g., there is no person the user can talk with about the system.

<u>PROCEDURE</u>:

<u>teaching method</u>:

Lecture/discussion on initial concepts was followed by working through the Preliminary Client Request for a video rental system. A discussion of the completeness and consistency of the client request was followed by a discussion of a Systems Requirement List. This can also include a small group exercise. In discussing the requirements, lead the students toward the need for testable requirements.

vocabulary introduced:

- customer(has the money)
- problem space
- problem specification
- requirements
- software requirements specification(SRS)
- viewpoints (customer, user, etc.)

INSTRUCTIONAL MATERIALS:

overheads:

L3OH1	Preliminary requirements list - KoFF video rental system

handouts:

L3HD1	Preliminary client request-video rental system

RELATED LEARNING ACTIVITIES:
(labs and exercises)

LAB001

CI 1, for small project requirements list is an exercise on this subject.

The KoFF preliminary client request for a video rental system can be used as an in-class or take home exercise. Have the students fill in the missing details in the narrative and discuss the results in class; then have them develop a complete requirements list from their revised client request.

READING ASSIGNMENTS:

Sommerville Chapter 3 (pp. 47-63)
Mynatt Chapter 2 (pp. 44-49) and (pp. 70-74)

RELATED READINGS:

Berzins Chapter 2 (pp. 23-75)
Ghezzi Chapter 2 (pp. 20-29)
Pressman Chapter 6 (pp. 173-177)
Schach Chapter 6 (pp. 137-153)

KoFF Preliminary System Requirements

1 KoFF shall accept membership application information which includes name, address, social security number and charge card information.

2 KoFF shall validate charge cards and generate unique RRR club numbers, both of which shall be recorded along with the membership applications.

3 KoFF shall charge applicants the membership fee.

4 KoFF shall, upon request from a current club member, display a list of available video tapes.

5 KoFF shall verify that member's card has not expired.

6 KoFF shall dispense the selected video tape to a valid club member.

7 KoFF shall bill the customer the fee for the dispensed video tape and retain the membership card.

8 KoFF shall accept returned video tapes and return the membership card after billing for any late fees.

9 KoFF shall make automated phone calls when video tapes are five days late.

10 KoFF shall void all cards when a tape is ten days late and make appropriate charges.

11 KoFF shall print membership cancellation letters.

12 KoFF shall dispense sale tapes and issue a charge to customer's account.

13 KoFF shall print rental tracking information every two weeks.

14 KoFF shall print membership information on request.

Automated Video Rental System Description

Client Request

Mr. Richard wants a computerized automated video cassette rental system which will be housed in unstaffed kiosks. These kiosks can be free standing in mall parking lots or can be placed in enclosed shopping malls. This device, KoFF (Kiosk of Famous Flicks), will accept applications for membership in Mr. Richard's Rapid Rental club (RRR), display titles of available tapes, dispense tapes, accept returned tapes, and take care of billings. It will also maintain reports of rental transactions.

One becomes a member of the club by entering membership information on a keyboard attached to the kiosk. This information will include a current charge card number and an approval to automatically charge that card for selected items including a membership fee of $ 10.00. Customers will be notified of membership in RRR by mail and will receive three RRR movie rental cards and a unique personal identification number. Membership expires on the expiration date of their charge card.

The kiosk contains 250 different tape titles and 1380 individual tapes. A customer can see a list of the available tapes by category by inserting one of their membership cards into the kiosk. The customer can select an available tape and rental duration. They will be charged for it and the tape will be dispensed from the tape out slot. Their card will be retained until the tape is returned to that kiosk. When a tape is returned to the tape-in slot, its bar code will be scanned, the customer will automatically be charged appropriate late fees and the membership card will be returned. Failure to return the tape within five days of its due date generates a phone call to the customer which plays a recorded message about the overdue tape and the accruing late charges. When the 10-day late limit is reached, the customer is charged for the late days and the cost of the tape. The customer is also charged a tape restocking fee and all of his/her membership cards are invalidated. The customer is notified of these actions.

The selection of videos must be updated. KoFF keeps information to help in this process. Videos which have not been rented for two weeks are listed for removal and videos which have been rented several times in a week are listed for additional copies. Every two weeks KoFF sends Mr. Richard's computer a copy of this report. He decides which tapes to add and which to remove. He updates the list of titles and records the quantities of those titles along with their identifying bar codes. He also assigns the rental price for that title. Sometimes instead of replacing a slow moving tape, he simply drops its rental price or tries to sell it. Sale tapes are indicated on a special screen. When a customer selects a sale tape, a record of the sale is made and the tape is dispensed.

Mr. Richard gets several reports from KoFF, including lists of sold tapes, the rental activity of RRR members by tape title and tape category -- Adventure, Comedy, Children, Restricted, the rental activity of particular titles and copies of that title, and detailed and summary financial reports of RRR member accounts.

KoFF Preliminary System Requirements List

1. KoFF shall accept membership application information which includes name, address, social security number and charge card information.
2. KoFF shall validate charge cards and generate unique RRR club numbers, both of which shall be recorded along with the membership applications.
3. KoFF shall charge applicants the membership fee.
4. KoFF shall, upon request from a current club member, display a list of available video tapes.
5. KoFF shall verify that member's card has not expired.
6. KoFF shall dispense the selected video tape to a valid club member.
7. KoFF shall bill the customer the fee for the dispensed video tape and retain the membership card.
8. KoFF shall accept returned video tapes and return the membership card after billing for any late fees.
9. KoFF shall make automated phone calls when video tapes are five days late.
10. KoFF shall void all cards when a tape is ten days late and make appropriate charges.
11. KoFF shall print membership cancellation letters.
12. KoFF shall dispense sale tapes and issue a charge to customer's account.
13. KoFF shall print rental tracking information every two weeks.
14. KoFF shall print membership information on request.

KoFF Preliminary System Requirements (instructor's notes)

This exercise is designed to illustrate the difficulty of abstracting requirements. It shows the importance of iteration. Several of the items in the requirements list above are incomplete. They do not meet even the explicit conditions of the customer request. Some of the missing requirements are listed below. A good exercise is to have the students fill in the missing requirements. Several of the requirements in the list are deliberately ambiguous and others are vague. Have the students resolve the ambiguities and remove the vagueness. Notice that none of the requirements talk about response time for example. There are several unstated requirements of the system. The client request does not even deal with how a customer can renew their membership. There is some opportunity for a discussion of professionalism, because the question of exception conditions is not even touched in the client request and not addressed in the requirements list. A discussion about the professional's responsibility to produce a quality system is useful here. What is the professional's responsibility to help design a more effective system? The requirements do not address non-system problems such as damaged tapes nor do they address detection of fraud such as the return of empty tape boxes or the return of tapes boxes with blank tapes in them.

When developing DFDs and Structure charts this exercise can be easily partitioned into customer management, tape management, financial management and system reporting segments. The concept of design partitioning can be related to design modularity in later discussions.

The example also provides an opportunity for the discussion of some ethical issues. RRR members are having detailed information about their rental habits retained by the system. Is this a violation of their privacy rights? The system does not need to associate the member's names with the rental of a video. The management of the system only requires capturing information about the frequency of rental of a particular video. Is capture of this information consistent with the ACM Code of Ethics and Professional Conduct section II?

The requirements list below includes some of the missing requirements.

1. KoFF shall accept membership application information which includes name, address, social security number and charge card information.

   A telephone number is also needed to call delinquent accounts.

2. KoFF shall validate charge cards and generate a unique RRR club numbers, both of which shall be recorded along with the membership applications.

   KoFF shall record the expiration date and other charge card information.

   KoFF shall produce three membership cards with membership information and print letters of acceptance for new members. or KoFF shall print the order to make and mail the cards and acceptance letter.

(How does KoFF process applications from members who have been rejected for not returning tapes within the ten-day grace period?)

3. KoFF shall charge applicants the membership fee.

4. KoFF shall, upon request from a current club member, display a list of available tapes and their rental price.

5. KoFF shall verify that member's card has not expired.

   KoFF shall do ?what? if the card is expired.

6. KoFF shall dispense selected tape to a valid club member.
   (How are tapes stuck in the dispensing chute handled?)

   KoFF shall require the selection of a rental (a charge) duration.

7. KoFF shall bill the customer the fee for that video and retain the membership card.

8. KoFF shall accept returned tapes and return the membership card after billing for any late fees.

   provided the card has not expired during the rental.

   KoFF shall not dispense tapes to members whose cards are within 10 days of expiration. (If card expires during rental period then there is no way to charge for an unreturned tape.)

9. KoFF shall make automated phone calls when tapes are five days late. (What is the content of that call?)

10. KoFF shall void all cards when a tape is ten days late (Is this count based on ten 24 hour units or on ten calendar days?) and make appropriate charges. (Because cards are voided, no one can access the system to return a tape which is eleven days late. What happens if they put the tape in the tape-in slot? Does the system keep the tape it already charged the customer for, or does it return the tape to the customer?)

    KoFF shall capture all voided cards when they are entered.

11. KoFF shall print membership cancellation letters.

12. KoFF shall dispense sale tapes and issue a charge to customers account.

    KoFF shall return the card with the sale tapes.

    KoFF shall display sale tapes when requested by a member.

13. KoFF shall print rental tracking information every two weeks.

14. KoFF shall print membership information on request.

LECTURE NUMBER: 004

TOPIC(S) FOR LECTURE:

Introduction to the structured analysis model.

INSTRUCTIONAL OBJECTIVES:

1. Understand the concept, notation, and relationships between:

    a) context diagram,

    b) data flow diagrams, and

    c) data dictionary.

2. *Construct a context diagram from a narrative description of a system.*

3. Construct a first-level data flow diagram from a narrative description of a system and a context diagram.

4. Construct data dictionary items for the data flows and data stores in the context diagram and data flow diagrams.

5. Understand the concepts of leveling and balancing in data flow diagrams.

SET UP,WARM-UP:

(How involve learner: recall, review, relate)

Recall our earlier discussions of the various activities/phases involved in software development and your work on developing a requirements list for your projects. What you've been involved in is defining the problem to be solved. Requirements analysis and specification (or just analysis) involves defining the problem. In general, analysts ask "what" type of questions; what is the problem to be solved; what is needed; what do you want the system to do. Analysts extract requirements and then they specify them (write them down). Common sense tells us that we have to define a problem before we can solve it; that forging ahead without fully understanding the problem isn't an effective approach, particularly with complex problems. Only when the problem has been analyzed (defined and specified) does it make sense to start considering how to solve it, i.e., consider design. Designers ask "how" type of questions; how are we going to solve the problem.

(Learning Label- Today we are going to learn ...)

Today we're going to look at some methods for modeling a system in order to understand it and to develop and clarify requirements. Specifically we're going to look at structured analysis.

CONTENTS:

1. Hand out narrative description of small college book ordering example. L4HD1

    a. Give class a few minutes to read it.

    b. Suggest visualizing system "in action" and imagine what (physical) inventory cards, department book requests, books needed file, book order form, and order list might look like. Show overheads of these to clarify and assure that everyone understands the system.

i	Inventory card	L4OH1
ii	Book request	L4OH2
iii	Books needed file	L4OH3
iv	Book order form	L4OH4
v	Order list	L4OH5

2. Context diagram (CD)
L4OH6
Use CD for small college book ordering example to introduce purpose, concept, notation, and vocabulary related to CDs.

    a. CD is the first level of the structured analysis model.

    b. CD defines system scope; boundaries.

    c. CD shows net flows of information into and out of the system. The notation for a net flow is a vector pointing in the direction of the flow.

    d. CD shows external entities (source,sink); things outside the system with which the system must interact. The notation for an external entity is a rectangle. Mynatt calls a context diagram a high-level data flow diagram.

3. Data dictionary (DD)
   L4OH7
   Use DD for small college book ordering example to introduce purpose, concept, notation, and vocabulary related to DDs.

   a. All data flows in CD will be described in the data dictionary. As with word entries in a normal dictionary, DD items are arranged in an easily retrievable order (alphabetical) and provide a detailed definition of the item.

   b. Discuss entries from the example to explain notation and relationship between the CD and DD.

   c. Review Mynatt's DD conventions. L4OH8

4. Based on sample CD and DD, recap how CD and integrated DD convey items above. Note that the CD views the system from the outside.

5. Data Flow Diagrams (DFDs)

   a. Now that we understand the boundaries of the system and how it interacts with external entities, we can look inside the system. That is, we can begin considering what is needed to transform the net inputs into the net outputs. The next level of modeling is the first-level DFD.

   L4OH9
   b. Use DFD for small college book ordering example to introduce DFD purpose, concept, and notation, and the relationship between the CD, DD, and DFD.

      i Transform (process, function) - transforms input flows into output flows. Each transform name should describe the purpose of the transform and consist of an action verb and object. The notation for a transform is an oval, circle, or rounded rectangle.
      ii Data flow - data in motion. Each data flow must appear in the data dictionary. Each data flow must be labeled unless it is going to or from a data store and the label would be the same as the data store name. Dataflow names are always nouns.
      iii Data store - data at rest. Data repository; place where data stored; represents a time delay. Each data store must also appear in the data dictionary. Data store names are always nouns.

iv Very briefly introduce the concept of leveling by suggesting that we could focus on a particular transform of the first-level DFD and draw another DFD (a child diagram) representing what goes on inside that transform. Introduce parent-child diagram concept.

v It is important to adopt some consistent naming and numbering notation in order to easily move between different levels of DFDs. Describe convention for numbering diagrams and transforms in diagrams.

vi A transform that cannot to be broken down any further is called a primitive transform.

c. There are two methods to get a first draft of a first-level DFD using a context diagram:

i create an event list (an event is something to which the system must respond). For each event, construct a transform representing the system's response to the event; then connect the transforms adding appropriate internal data flows and data stores.

ii construct a transform to receive each of its input data flows and a transform to produce each output data flow.

Note that these are simply ways to get started by identifying transforms. Use the context diagram L4OH6 to illustrate method ii. You should derive a DFD with four transforms: Get department requests, Buy used books, Receive new books, and Generate book order form. This model is not yet complete and does not model the problem. You need to refine this model in several ways. Possible refinements include adding some internal data flows and data stores to allow transforms to interface properly, combining, adding or eliminating transform to more accurately reflect the system. The result will look something like L4OH9.

d. Illustrate the concept of leveling and establishment of a consistent numbering/naming convention, by decomposing the replenish books transform. Discuss the parent/child relationship of CD and different levels of DDs.

e. Illustrate and give a brief introduction to the concept of balancing between DFD levels, including both data balancing and functional balancing.

6. Physical model vs logical model

a. Physical model - implementation dependent; useful in depicting existing system. Point out that our model of the small college book ordering system is a physical model.

b. Logical model - implementation independent; useful in requirements analysis and specification. Point out that during analysis we want to avoid implementation details and develop a logical model of the system.

PROCEDURE:

teaching method:

The small college book ordering example is used to introduce the concepts, notation, and vocabulary of context diagrams, data flow diagrams, and data dictionary. The class is given a few minutes to familiarize themselves with the requirements followed by a short discussion to assure that they understand the system and can visualize it in action. A CD, then a DD, and finally a first-level DFD for the system are provided and explained.

vocabulary introduced:

structured analysis
context diagram
external entities, source, sink
data flows
data dictionary
data flow diagrams
transform, process, activity
data store
leveling
parent/child diagrams
balancing
data balancing
functional balancing
event, event list
physical model, logical model

INSTRUCTIONAL MATERIALS:

overheads

L4OH1	Small College book ordering: Inventory card
L4OH2	Small College book ordering: Book request
L4OH3	Small college book ordering: Books needed file
L4OH4	Small College book ordering: Book order form
L4OH5	Small College book ordering. Order list
L4OH6	Context diagram - Book Order System
L4OH7	Data dictionary notation examples
L4OH8	Data dictionary conventions

L4OH9	1st Level DFD - Book Order System

<u>handouts</u>

L4HD1	Small college textbook ordering example

<u>RELATED LEARNING ACTIVITIES:</u>
(labs and exercises)

The small college book ordering system can be used for in-class and lab exercises to reinforce concepts, vocabulary, and notation for CDs, DDs, and DFDs.

Lab002 is an exercise on CDs and first level DFDs.

<u>READING ASSIGNMENTS</u>:
Mynatt Chapter 4 (pp. 44-62)

<u>RELATED READINGS</u>:
Berzins Chapter 3 (pp. 109-112)
Ghezzi Chapter 5 (p. 161)
Pressman Chapter 7 (pp. 208-211)
Schach Chapter 7 (pp. 162-170)

EXAMPLE - SMALL COLLEGE BOOK ORDERING

The following describes how the bookstore at a small private college manages the ordering of textbooks.

The bookstore maintains an inventory card for each course in the college catalog. Each inventory card contains the title, author, and publisher of the textbook currently used. It also contains the number of the textbooks that are already in stock.

Midway through the spring semester, each academic department provides the bookstore with textbook information for each course they will be offering in the next academic year. The information provided is title, author, and publisher of textbook to be used, and the expected course enrollment, if known.

The bookstore then creates a Books Needed File containing the title, author, publisher, and number needed (expected enrollment minus the number in stock) for each book to be used in the next academic year.

During the last week of the spring semester the bookstore will buy books from students if the Books-Needed-File indicates a need. Of course, each time a used book is purchased, appropriate updates are made in the bookstore's records.

Over the summer the bookstore prepares an Order-List containing the title, author, publisher, and number to be ordered for each book that is still needed. The Order List is then used to create an individual Book Order Form for each publisher. These Book Order Forms are sent to the publishers.

Small College Book Ordering System

Inventory Card

Course:

Textbook title:

Author:

Publisher:

Number in Stock:

Small College Book Ordering System

Book Request

Course:

Textbook Title:

Author:

Publisher:

Expected Enrollment:

Small College Book Ordering System

BOOKS NEEDED FILE

BOOK TITLE	AUTHOR	PUBLISHER	NUMBER NEEDED
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

{one entry for book to be used next year}

Small College Book Ordering System

ORDER LIST

Book Title	Author	Publisher	Qty
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

{one entry for each book to be ordered}

Small College Book Ordering System

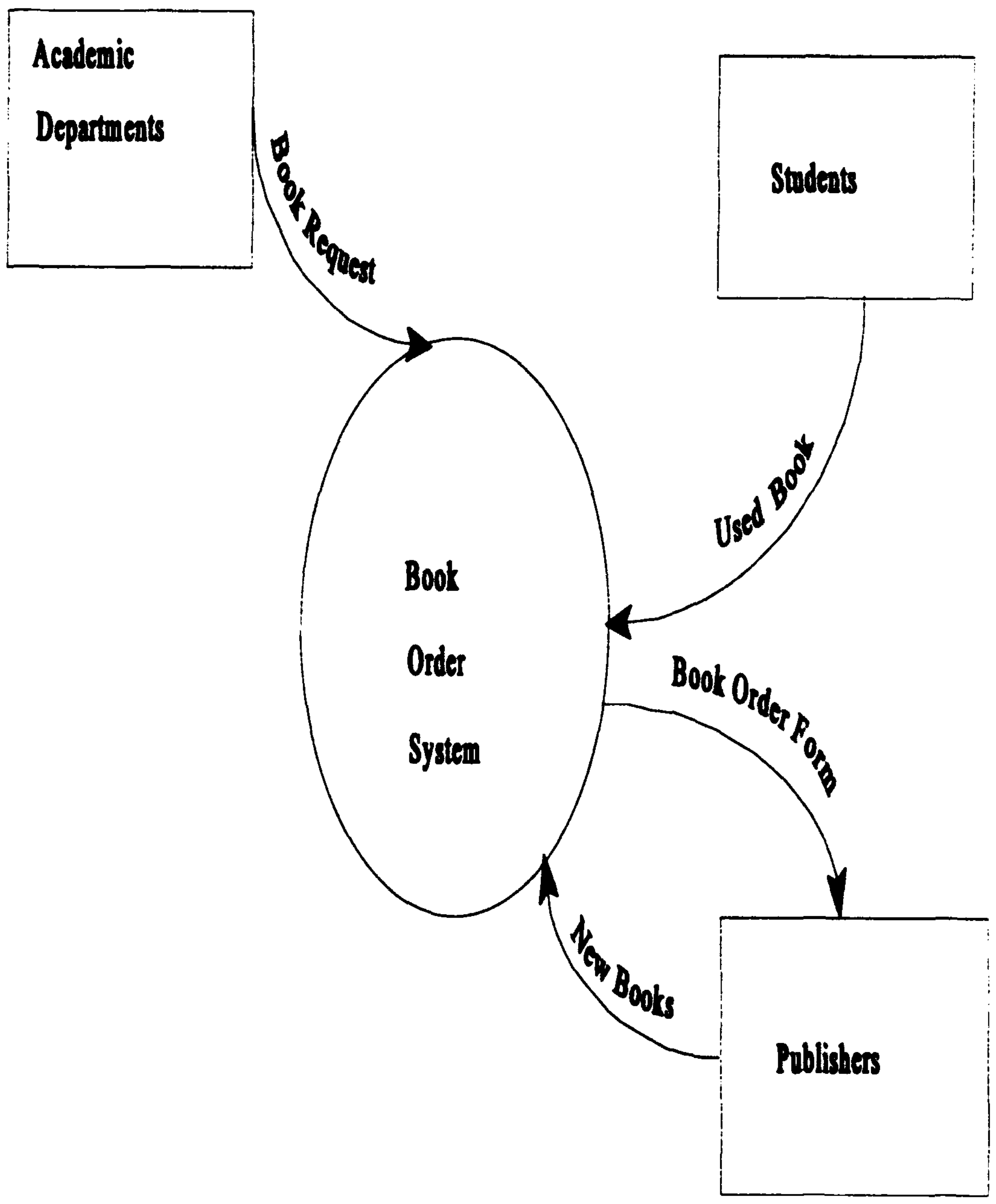
BOOK ORDER FORM

Publisher:

<u>Book Title</u>	<u>Author</u>	<u>Quantity</u>
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.

Context Diagram

Book Order System



Data Dictionary Notation Examples

Book Request = Course + Title + Author + Publisher + [Expected Enrollment]

Course = Department + Course Number

Course Number = 1|2|3| + { digit }$_3$

Digit = 1|2|3|4|5|6|7|8|9|0

Department = School Code + Department Number

Inventory Card = Title + Author + Publisher + InStock

An inventory card is maintained for each course in the college catalog.

Inventory File = { Inventory Card }

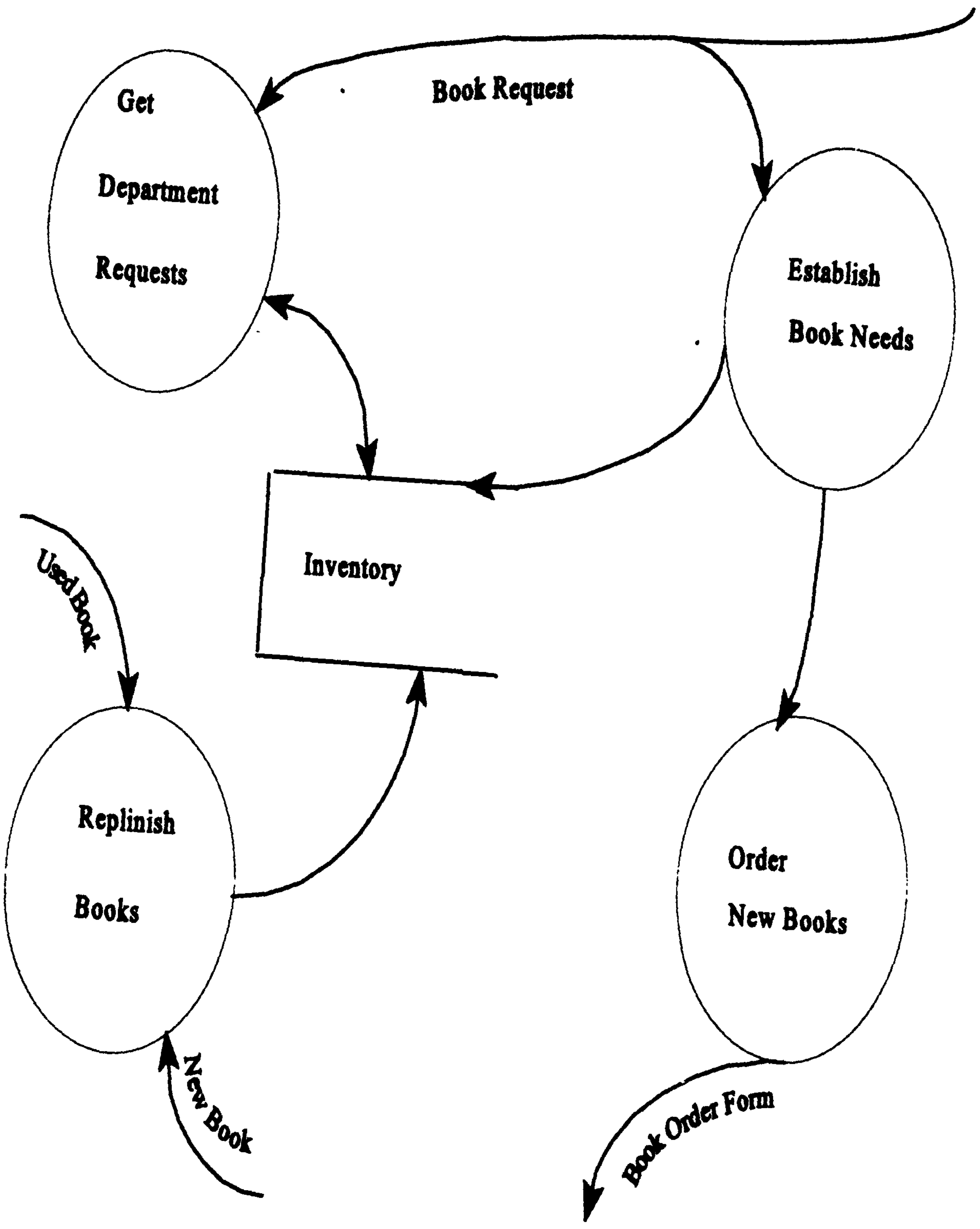
Order List ={Title + Author + Publisher + Quantity}

Data Dictionary Conventions

=	is equivalent to / is comprised of
+	AND / together with
|	either-or
[ ]	one or more optional elements
{ }	iterations of
	$\{\ \}^{*}$ upper limit
	$\{\ \}_{*}$ lower limit
' '	literals

Comments may be added to a data dictionary.

1st Level DFD
Book Order System



L4OH9

<u>LECTURE NUMBER</u>: 005

<u>TOPIC(S) FOR LECTURE</u>:

Quality standards in requirements
Requirements extraction using the video rental example
Development of DFDs using an analysis of system inputs and outputs
Balancing and data dictionaries

<u>INSTRUCTIONAL OBJECTIVE(S)</u>:

1. Recognize and correct problems in a requirements list.

2. Develop DFDs from a requirements list.

3. Determine if DFDs are balanced.

<u>SET UP, WARM-UP</u>:
(How involve learner: recall, review, relate)

As you learned in your laboratory requirements project (Lab001) to develop CI-1, it is not easy to generate a complete and effective requirements statement. (Display the preliminary client request.) L3HD1
What sort of thing does this request fail to do? (Discuss some of the obvious failings of the request, such as, the failure to get a client phone number or the failure to say what should be done when clients insert an expired card into the system).

(Learning Label- Today we are going to learn ...)

Today we are going to learn how applying some standards of quality requirements helps to clarify the desired functionality of a system and how we can move from a requirements list to data flow diagrams.

<u>CONTENTS</u>:

1. Present standards for quality requirements. Refer to failure of the Preliminary Video Rental (KoFF) Client Request to meet these standards.

   a. Requirements should be testable.

   b. Requirements should be specific.

   c. Requirements should be feasible.

L5HD1

2. Handout and display the Revised Client Request and work through each of the changes characterizing why they were made and how they relate to the above standards.

   a. The addition of a time limit for transactions introduces specificity and testability. However, the time limit cannot apply to things outside the system. This presages the use of DFDs and context diagrams to help clarify what is outside of the system.

   b. The addition of charge card type and date is needed for the system to later validate the card.check later.

   c. The inclusion of a requirement for more detailed customer information leads to a more specific testable requirement.

   d. Because it is not feasible to print reports at the kiosk, report information must be transmitted outside the kiosk.

   e. This is a good time to consider whether the software developer is responsible for leaving something out of the system that the user did not disclose to him. In some cases the developer would not otherwise have any knowledge that an item would be needed and in other cases the developed might know of useful additions to the system. Briefly discuss ethics issues and the developer's respons*ibility here.*

L5HD2

3. Display the revised requirements list showing how the changes in the Client Request are reflected in the requirements list.

   a. A good example of ambiguity is the 10-day late penalty. Until the word calendar was added to requirement 10 from the preliminary requirements(L3OH1) (Now requirement 15) it was not clear whether the requirement referred to calendar dates or 24-hour clock periods. Note that this problem is pervasive. Refer to additional examples, e.g. Sommerville reading.

4. How do you generate DFDs from a requirements list?

   L5OH1

   a. Also discuss the inputs and outputs to the system and develop a list from the students; then show overhead

   L5OH2

b. Using the input-output list show them how to generate a context diagram(Mynatt). Do only part of the list on the board and then show the complete context diagram. Relate each of the inputs and outputs back to the requirements list. Note how some inputs may be merged under a single label and this is only revealed by use of a data dictionary. Membership application is a good example of this.

L5OH3

c. Talk about the major functions of the system -membership control, tape control, billing, and report generation - as the initial transforms of the DFD. Show the level one DFD.

L5OH4

d. Show the next level of the Manage Membership transform as an example of leveling.

e. Revisit the concept of balancing
   - i Ask why the inputs and outputs for member management in diagram 1 do not match the inputs and outputs for member management in diagram 0. This shows that the verification of balancing depends on the data dictionary.
   - ii Reintroduce the concept of a data dictionary showing them the entry for "Application" L5OH5 which consists of several elements.
   - iii Explain to students that there are several reasons for dividing up complex data flows. These reasons include reference to *some elements* which may be classified or privileged information, or some other process may only look at one element

<u>PROCEDURE</u>:

<u>teaching method</u>:

Lecture on initial concepts is followed by working through the Preliminary Client Request for a video rental system. An interactive discussion of the completeness and consistency of the client request is followed by a discussion and development of DFDs.

<u>vocabulary introduced</u>:

feasible
leveling
testable
specificity

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>:

L3HD1	Preliminary client request-video rental system
L5OH1	System input and output list
L5OH2	Context diagram-video rental system
L5OH3	Level zero DFD-video rental system (Manage Membership)
L5OH4	Level one DFD - video rental system (Manage Membership)
L5OH5	Example data dictionary entry from KoFF DD

<u>handouts</u>:

L5HD1	Revised client request for KoFF automated video rental system
L5HD2	Adjusted requirements list for KoFF video rental system

<u>RELATED LEARNING ACTIVITIES</u>:
(labs and exercises)

Lab003 - Building the CD, DFDs, and DD for the small project is an exercise on this subject.

<u>READING ASSIGNMENTS:</u>

Sommerville Chapter 3 (pp. 47-63)
Mynatt Chapter 2 (pp. 44-62)

Automated Video Rental System Description

Client Request

Mr. Richard wants a computerized automated video cassette rental system which will be housed in unmanned kiosks. These kiosks can be free standing in mall parking lots or can be placed in enclosed shopping malls. This device, KoFF (Kiosk of Famous Flicks), will accept applications for membership in Mr. Richard's Rapid Rental club (RRR), display titles of available tapes, dispense tapes, accept returned tapes, and take care of billings. It will also maintain reports of rental transactions.

One becomes a member of the club by entering membership information on a keyboard attached to the kiosk. This information will include a current charge card number and an approval to automatically charge that card for selected items including a membership fee of $ 10.00. Customers will be notified of membership in RRR by mail and will receive three RRR movie rental cards and a unique personal identification number. Membership expires on the expiration date of their charge card.

The kiosk contains 250 different tape titles and 1380 individual tapes. A customer can see a list of the available tapes by category by inserting one of their membership cards into the kiosk. The customer can select an available tape and rental duration. They will be charged for it and the tape will be dispensed from the tape out slot. Their card will be retained until the tape is returned to that kiosk. When a tape is returned to the tape-in slot, its bar code will be scanned, the customer will automatically be charged appropriate late fees and the membership card will be returned. Failure to return the tape within five days of its due date generates a phone call to the customer which plays a recorded message about the overdue tape and the accruing late charges. When the 10-day late limit is reached, the customer is charged for the late days and the cost of the tape. The customer is also charged a tape restocking fee and all of his/her membership cards are invalidated. The customer is notified of these actions.

The selection of videos must be updated. KoFF keeps information to help in this process. Videos which have not been rented for two weeks are listed for removal and videos which have been rented several times in a week are listed for additional copies. Every two weeks KoFF sends Mr. Richard's computer a copy of this report. He decides which tapes to add and which to remove. He updates the list of titles and records the quantities of those titles along with their identifying bar codes. He also assigns the rental price for that title. Sometimes instead of replacing a slow moving tape, he simply drops its rental price or tries to sell it. Sale tapes are indicated on a special screen. When a customer selects a sale tape, a record of the sale is made and the tape is dispensed.

Mr. Richard gets several reports from KoFF, including lists of sold tapes, the rental activity of RRR members by tape title and tape category -- Adventure, Comedy, Children, Restricted, the rental activity of particular titles and copies of that title, and detailed and summary financial reports of RRR member accounts.

KoFF Preliminary System Requirements List

1. KoFF shall accept membership application information which includes name, address, social security number and charge card information.

2. KoFF shall validate charge cards and generate unique RRR club numbers, both of which shall be recorded along with the membership applications.

3. KoFF shall charge applicants the membership fee.

4. KoFF shall, upon request from a current club member, display a list of available video tapes.

5. KoFF shall verify that member's card has not expired.

6. KoFF shall dispense the selected video tape to a valid club member.

7. KoFF shall bill the customer the fee for the dispensed video tape and retain the membership card.

8. KoFF shall accept returned video tapes and return the membership card after billing for any late fees.

9. KoFF shall make automated phone calls when video tapes are five days late.

10. KoFF shall void all cards when a tape is ten days late and make appropriate charges.

11. KoFF shall print membership cancellation letters.

12. KoFF shall dispense sale tapes and issue a charge to customers account.

13. KoFF shall print rental tracking information every two weeks.

14. KoFF shall print membership information on request.

KoFF Preliminary System Requirements (Instructor's notes)

This exercise is designed to illustrate the difficulty of abstracting requirements. It shows the importance of iteration. Several of the items in the requirements list above are incomplete. They do not meet even the explicit conditions of the customer request. Some of the missing requirements are listed below. A good exercise is to have the students fill in the missing requirements. Several of the requirements in the list are deliberately ambiguous and others are vague. Have the students resolve the ambiguities and remove the vagueness. Notice that none of the requirements talk about response time for example. There are several unstated requirements of the system. The client request does not even deal with how a customer can renew their membership. There is some opportunity for a discussion of professionalism, because the question of exception conditions is not even touched in the client request and not addressed in the requirements list. A discussion about the professional's responsibility to produce a quality system is useful here. What is the professional's responsibility to help design a more effective system? The requirements do not address non-system problems such as damaged tapes nor do they address detection of fraud such as the return of empty tape boxes or the return of tapes boxes with blank tapes in them.

When developing DFDs and Structure charts this exercise can be easily partitioned into customer management, tape management, financial management and system reporting segments. The concept of design partitioning can be related to design modularity in later discussions.

The example also provides an opportunity for the discussion of some ethical issues. RRR members are having detailed information about their rental habits retained by the system. Is this a violation of their privacy rights? The system does not need to associate the member's names with the rental of a video. The management of the system only requires capturing information about the frequency of rental of a particular video. Is capture of this information consistent with the ACM Code of Ethics and Professional Conduct section II?

The requirements list below includes some of the missing requirements.

1. KoFF shall accept membership application information which includes name, address, social security number and charge card information.

   A telephone number is also needed to call delinquent accounts.

2. KoFF shall validate charge cards and generate a unique RRR club numbers, both of which shall be recorded along with the membership applications.

   KoFF shall record the expiration date and other charge card information.

   KoFF shall produce three membership cards with membership information and print letters of acceptance for new members. or KoFF shall print the order to make and mail the cards and acceptance letter.
   (How does KoFF process applications from members who have been rejected for not

returning tapes within the ten-day grace period?)

3. KoFF shall charge applicants the membership fee.

4. KoFF shall, upon request from a current club member, display a list of available tapes and their rental price.

5. KoFF shall verify that member's card has not expired.

   KoFF shall do ?what? if the card is expired.

6. KoFF shall dispense selected tape to a valid club member.
   (How are tapes stuck in the dispensing chute handled?)

   KoFF shall require the selection of a rental (a charge) duration.

7. KoFF shall bill the customer the fee for that video and retain the membership card.

8. KoFF shall accept returned tapes and return the membership card after billing for any late fees.

   provided the card has not expired during the rental.

   KoFF shall not dispense tapes to members whose cards are within 10 days of expiration. (If card expires during rental period then there is no way to charge for an unreturned tape.)

9. KoFF shall make automated phone calls when tapes are five days late. (What is the content of that call?)

10. KoFF shall void all cards when a tape is ten days late (Is this count based on ten 24 hour units or on ten calendar days?) and make appropriate charges. (Because cards are voided, no one can access the system to return a tape which is eleven days late. What happens if they put the tape in the tape-in slot? Does the system keep the tape it already charged the customer for, or does it return the tape to the customer?)

    KoFF shall capture all voided cards when they are entered.

11. KoFF shall print membership cancellation letters.

12. KoFF shall dispense sale tapes and issue a charge to customers account.

    KoFF shall return the card with the sale tapes.

    KoFF shall display sale tapes when requested by a member.

13. KoFF shall print rental tracking information every two weeks.

14. KoFF shall print membership information on request.

DFD Preparation

System inputs:

- Member's name and address.
- Member's phone number.
- Member's charge card data.
- Membership card information.
- Membership card.
- Tape selection.
- Rental duration.
- Returned Tape

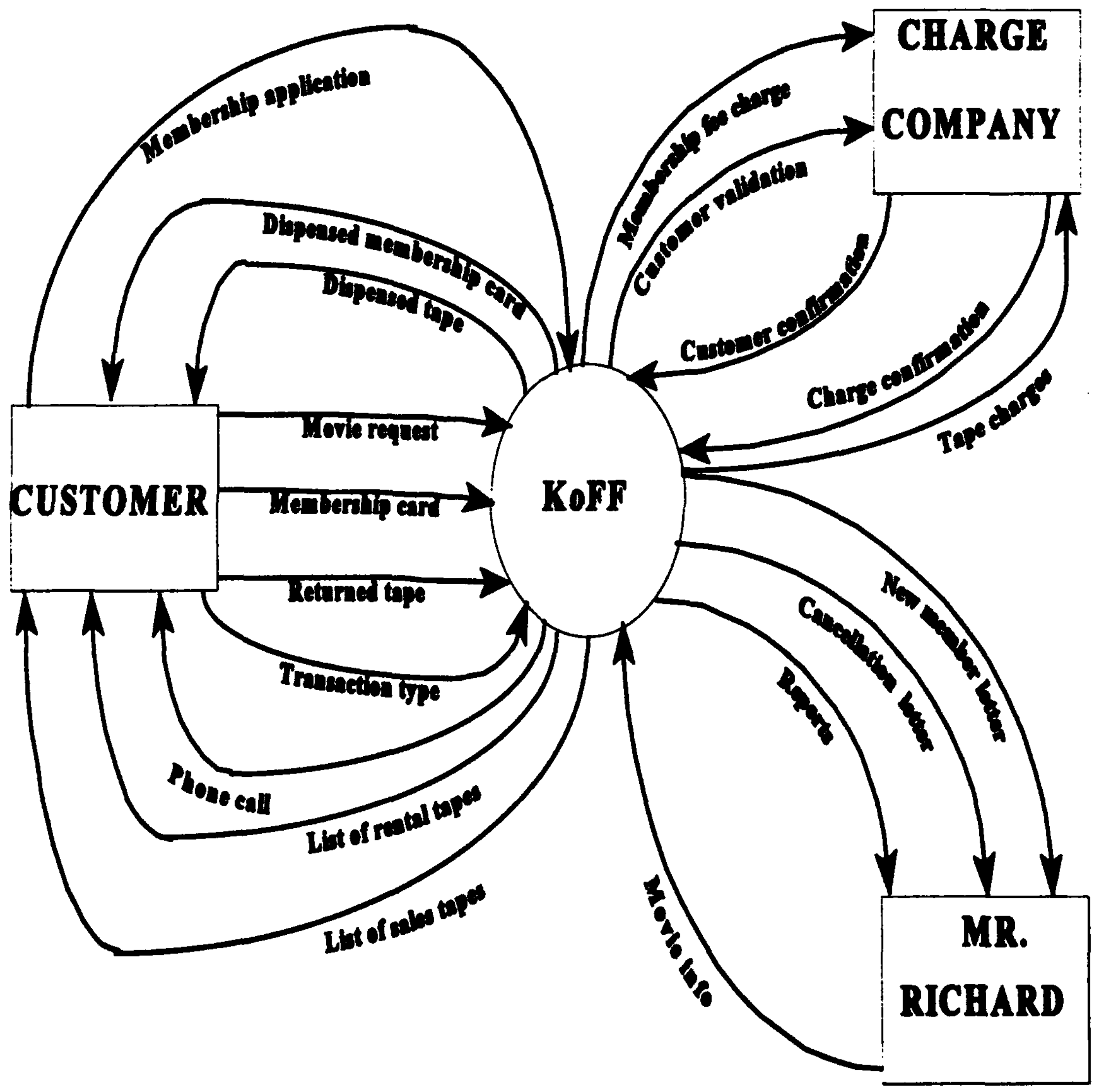
Internal processes:

- Generate card numbers.
- Validate card.
- Retain expired cards.
- Not process cards within 10 days of expiration.
- Void all cards of those who transgress the lateness limit.

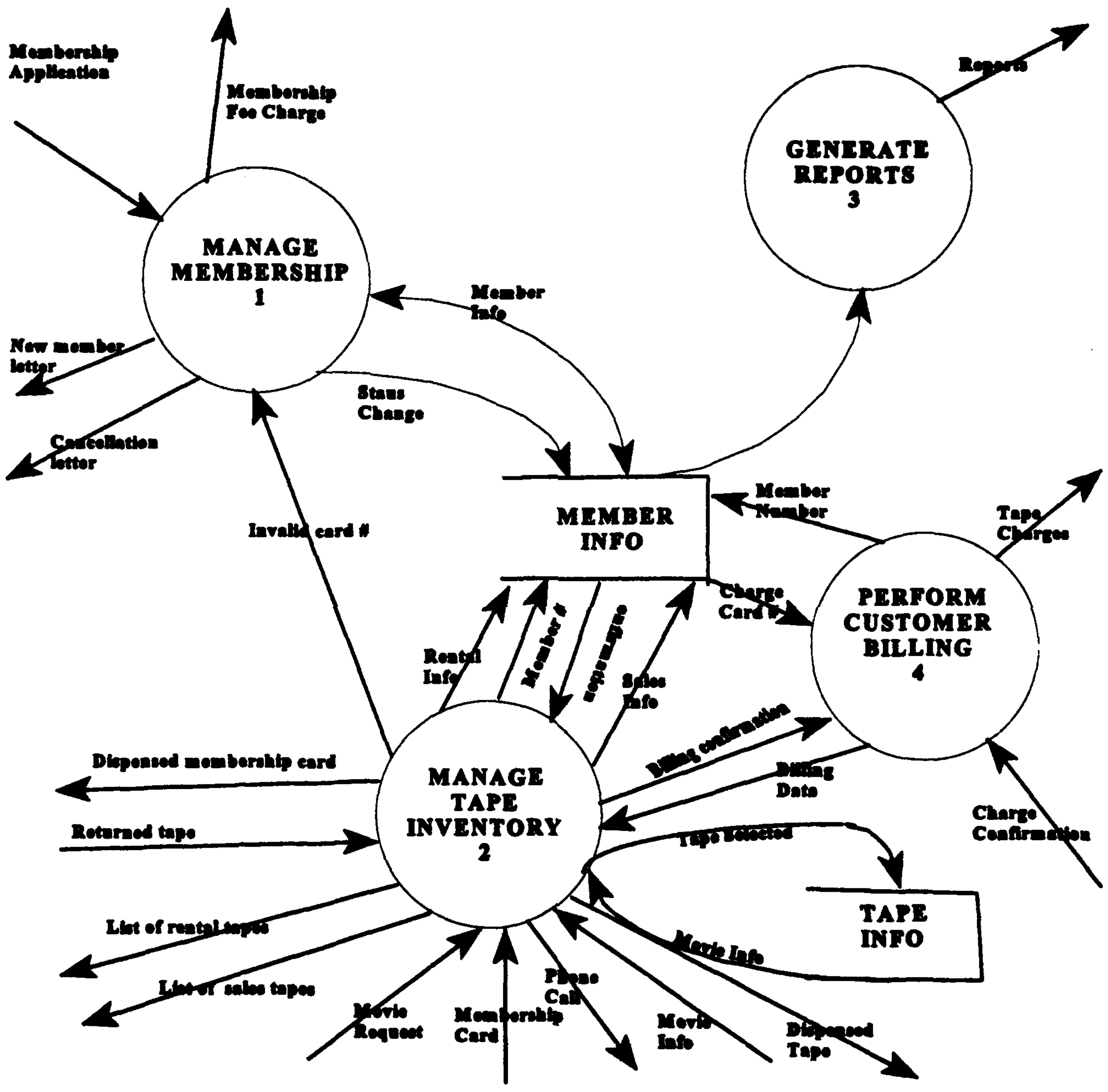
System outputs:

- Membership acceptance information.
- Membership billing to charge company.
- Available rental videos to the monitor.
- Videos for sale to the monitor.
- Sale and rental tapes.
- Membership card.
- Mr. Richard's financial reports.
- Dispensed tape
- Dunning phone call
- New member letter

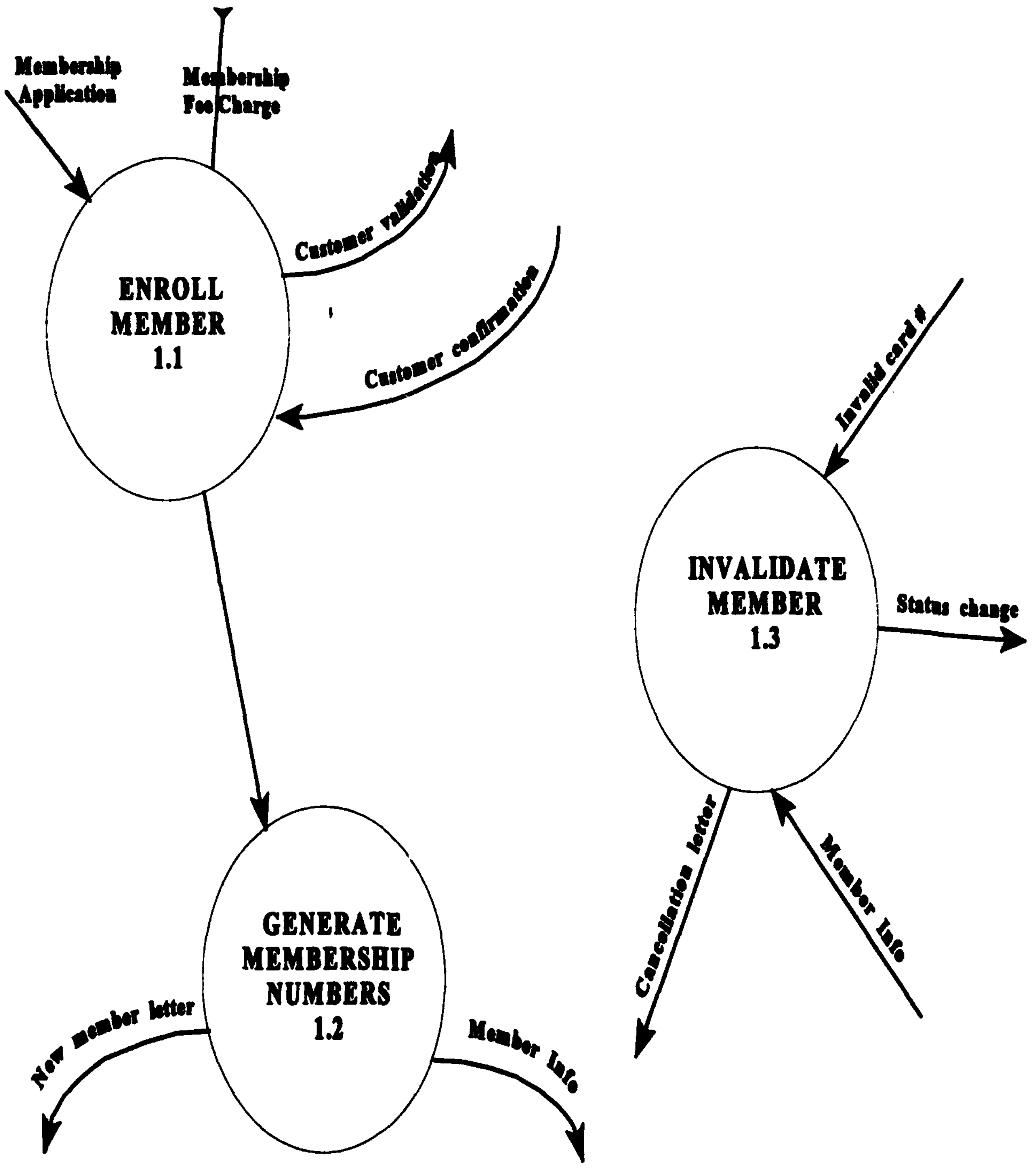
KoFF Automated Video Rental System Context Diagram



KoFF Automated Video Rental System
Diagram 0



KoFF Automated Video Rental System
Diagram 1



Excerpt from KoFF DD

Data Dictionary:

Application =	Name + Address + Phone number + Charge card type + Charge card number + Card Expiration Date.
Address =	Street-address + City-State + Zipcode.
Name =	First Name + Last Name.

Automated Video Rental System Description

REVISED* Client Request

Mr. Richard wants a computerized automated video cassette rental system which will be housed in unmanned kiosks. These kiosks can be free standing in mall parking lots or can be placed in enclosed shopping malls. This device, KoFF (Kiosk of Famous Flicks), will accept applications for membership in Mr. Richard's Rapid Rental club (RRR), display titles of available tapes, dispense tapes, accept returned tapes, and take care of billings. It will also maintain reports of rental transactions. **To assure customer satisfaction, all transactions with the customer should take place in less than 90 seconds.**

One becomes a member of the club by entering membership information on a keyboard attached to the kiosk. This information will include a current charge card number **type and expiration date** and an approval to automatically charge that card for selected items including a membership fee of $ 10.00. **Customer information will also include customer's name, mailing address and telephone number.** Customers will be notified of membership in RRR by mail **from Mr. Richard's office** and will receive three RRR movie rental cards and a unique personal identification number. Membership expires on the expiration date of their charge card.

The kiosk contains 250 different tape titles and 1380 individual tapes. A customer can see a list of the available tapes by category by inserting one of their **unexpired** membership cards into the kiosk. **Expired cards are captured by KoFF. Customers with more than 10 days to expiration can continue to interact with KoFF**. The customer can select an available tape and rental duration. They will be charged for it and the tape will be dispensed from the tape out slot. Their card will be retained until the tape is returned to that kiosk. When a tape is returned to the tape-in slot, its bar code will be scanned, the customer will automatically be charged appropriate late fees and the membership card will be returned. **There is a hardware device that determines if the correct tape was returned undamaged.** Failure to return the tape within five days of its due date generates a phone call to the customer which plays a recorded message about the overdue tape and the accruing late charges. When the 10-day late limit is reached, the customer is charged for the late days and the cost of the tape. The customer is also charged a tape restocking fee and all of his/her membership cards are invalidated. **KoFF shall transmit membership cancellation letters to Mr. Richard's computer. KoFF shall capture all invalidated cards.** The customer is notified of this.

The selection of videos must be updated. KoFF keeps information to help in this process. Videos which have not been rented for two weeks are listed for removal and videos which have been rented several times in a week are listed for additional copies. Every two weeks KoFF sends Mr. Richard's computer a copy of this report. He decides which tapes to add and which to remove. He updates the list of titles and records the quantities of those titles along with their identifying bar codes. He also assigns the rental price for that title. Sometimes instead of replacing a slow moving tape, he simply drops its rental price or tries to sell it. Sale tapes are indicated on a special screen. When a customer selects a sale tape, a record of the sale is made, the tape is dispensed **and the membership card is returned**.

Mr. Richard gets several reports from KoFF, including lists of sold tapes, the rental activity of RRR members by tape title and tape category ,i.e., Adventure, Comedy, Children, Restricted,

the rental activity of particular titles and copies of that title, and detailed and summary financial reports of RRR member accounts.

* Bold items reflect revisions

KoFF Adjusted Requirements list

1. KoFF shall accept membership application information which includes name, address, social security number and charge card information. **A telephone number is also needed to call delinquent accounts.**

2. KoFF shall validate charge cards and generate a unique RRR club numbers, both of which shall be recorded along with the membership applications.

3. **KoFF shall record the expiration date and other charge card information.**

4. **KoFF shall transmit the order to make and mail the cards and acceptance letter to Mr. Richard's computer.**

(How does KoFF process applications from members who have been rejected for not returning tapes within the ten-day grace period?)

5. KoFF shall charge applicants the **$10** membership fee.

6. KoFF shall, upon request from a current club member, display a list of available tapes and their rental price.

7. KoFF shall verify that a member's card has not expired.

8. **KoFF shall retain the card if the card is expired.**

9. KoFF shall dispense a selected tape to a valid club member.

(How are tapes stuck in the dispensing chute handled?)

10. **KoFF shall require the selection of a rental (a charge) duration.**

11. KoFF shall bill the customer the fee for the selected video and retain the membership card.

12. KoFF shall accept returned tapes and return the membership card after billing for any late fees, **provided the card has not expired during the rental.**

13. **KoFF shall not dispense tapes to members whose cards are within 10 days of expiration.** (If card expires during rental period then there is no way to charge for an unreturned tape.)

14. KoFF shall make automated phone calls when tapes are five days late. (What is the content of that call?)

15. KoFF shall void all cards and make appropriate charges when a tape is ten calendar days late. (Because cards are voided, no one can access the system to return a tape which is eleven days late.)

16. KoFF shall capture all voided cards when they are entered.

17. KoFF shall **transmit** membership cancellation letters **to Mr. Richard's computer.**

18. KoFF shall dispense sale tapes and issue a charge to customer's account within 1 minute of the start of the transaction.

19. **KoFF shall return the card with the sale tapes.**

20. **KoFF shall display sale tapes when requested by a member.**

21. KoFF shall **transmit** rental tracking information for two-weeks activity **when requested by Mr. Richard.**

22. KoFF shall **transmit** membership information upon request **from Mr. Richard**.

* Bold items reflect revisions.

Parentheses indicate questions for class discussion on potential weaknesses in the requirements.

LECTURE NUMBER: 006

TOPIC(S) FOR LECTURE:
Introduction to design

INSTRUCTIONAL OBJECTIVE(S):

1. Distinguish between analysis and design.
2. Identify key design goals.
3. Know the general inputs and outputs of design.
4. Understand the purpose and notation of structure charts.
5. Understand fan-in, fan-out, coupling, and cohesion as structured design criteria.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

During the first class we discussed the different activities in the software life cycle and some models of the life cycle. Some aspects of design have been touched upon in introductory programming classes, and possibly some other courses. Some areas of the topic will either be covered in more depth and other areas are new material.

We are taking a "spiral approach" to the material. Our first pass through some topics will be exactly that - a first pass. We intend the depth provided to be sufficient for application to your first project. Gaps will be filled in during subsequent passes in the spiral. Similarly there are many techniques and methodologies for analysis and design but we need to choose specific ones to apply to the first project in a timely fashion.

(Learning Label- Today we are going to learn ...)

L2OH8 L6OH1 L6OH2
In our second lecture we considered various activities of the software life cycle. The waterfall model shows the stages of software development. We have discussed requirements analysis and you have experienced this in your small projects. Today we are going to introduce the concept of design and how it relates to requirements.

CONTENTS:

1. What is design?

   a. Whereas analysis defines the problem space,i.e., what needs are to be met, design considers how to solve the problem; how to meet the requirement.

b. Design involves establishing the overall system architecture; the components and the relationship between the components

c. Identifies how to meet the specified requirements subject to the stated goals and constraints. Point out how customer goals (non-functional requirements) can lead to different solutions.

L6OH3

d. Preliminary design is the identification and selection of major system components and how they relate. (A black box view.)

L6OH4

e. Detailed design is a refinement of preliminary design in which the internal aspects of the components and the interfaces are detailed. (A white box view.)

f. In practice, we iterate between requirements analysis and design. Realistically there is often not the clean break between these activities that is implied by the life cycle. Typically it is not possible to completely specify the system and proceed to design knowing the requirements are stable.

L6OH5

g. Key design goals. Design is intended to solve the custopmers needs. Other important aspect of design that are frequently forgotten relate to both the developer and the customer. Software should be designed so that it is easily testable and has components that can potentially be revised. Designing with future maintenance in mind is also important.

2. Structure charts

a. Discuss the analogy of structure charts to architectural blueprints for a house. These are design documents; produced after the requirements have been determined. The blueprints show the architecture of the house: the components (rooms, heating system, plumbing, etc) and the relationship between the components.

b. Structure chart is one way to depict a software design.

L6OH6

Note notation and information conveyed:

i Components (modules) represented as rectangles.

ii Purpose of each component is conveyed in its name.

iii Interfaces between components (data couples, control couples) are show as vectors with labels. Data couples start with an open circle and end with an arrowhead. Control couples start with a filled-in circle and end with an arrow head. (Mynatt pg 152)

iv Hierarchy (make analogy with organization chart) showing who calls who, who reports to whom. As a familiar analogy, use segments of school's organization chart depicting President at level-1, and VP's at level-2. Take VP-Academic Affairs down to Dean level, chair level, and faculty level. As precursor to fan-in/fan-out, without using the terms, ask questions such as: What would you think about the organization if there were 37 vice-presidents reporting to the president? If there was 1 dean reporting to a VP? If there was a lower level function that was called upon by many different functions at higher levels?

3. Design measures/criteria (use L6OH6)

a. Fan-in - A component's fan-in is the number of higher level components that call upon it; its bosses. (Calculate Normal Deductions has fan-in of 2.)

b. Fan-out - A component's fan-out is the number of components that it calls upon its immediate subordinates. (Issue pay checks for all employees has a fan-out of 4.)

c. Coupling is a measure of dependency between components. A design goal is to minimize coupling by eliminating unnecessary dependencies. Intuitively, loosely coupled components are desirable because their independence makes them more maintainable and have a greater chance of being reusable.

d. Cohesion is a measure of the internal strength of a component; of how well the elements within a component contribute a single well-defined purpose. A design goal is to maximize cohesion. Intuitively, highly cohesive components are desirable. Analogies with familiar team and team concepts are useful here. For example, athletic teams that are cohesive (all of members work well together; often teams with lesser talent are more successful than teams with greater talent).

e. Note that strong cohesion and loose coupling are related; they have an inverse relationship. Minimizing coupling (the dependencies between components) will result in more cohesive components. Conversely, improving (increasing level of) cohesion will reduce (improve) coupling.

4. There are many different design strategies and methodologies: for example, structured design methods and object-oriented design methods. While we will be talking about some general design principles, we will use structured design in the first project and thus will be adopting some specific structured design notation and method.

PROCEDURE:

teaching method:
The intent at this point is to briefly introduce design in general and structured design in particular. A sample structure chart for a familiar type of system is used as a vehicle to describe the notation to be used and how to use it in design. Ask the students, near the end of the lecture, how the structured model helps in achieving the design goals identified in 1g above.

vocabulary introduced:
design
preliminary design
detailed design
general design goals (maintainability, reusability, ease of testing)
structured design
object-oriented design
structure chart
fan-in
fan-out
coupling
data couples
control couples
cohesion

INSTRUCTIONAL MATERIALS:

overheads:

L2OH8	Waterfall model
L6OH1	Software requirements analysis
L6OH2	Software specifications
L6OH3	Preliminary design
L6OH4	Detailed design
L6OH5	Key design goals
L6OH6	Example structure chart

handouts:

RELATED LEARNING ACTIVITIES:
(labs exercises)

Lab004	Discussion questions aimed at verifying understanding of the content and notation of structure charts are helpful. For example, provide a structure chart similar to that in OH5 and

ask about the hierarchy, interfaces and coupling between particular
modules, fan-in and fan-out for specific modules, and cohesiveness of specific modules.

<u>READING ASSIGNMENTS</u>:

Sommerville Chapter 10 (pp. 171-189)
Sommerville Chapter 12 (pp. 219-237)
Mynatt Chapter 4 (pp. 143-156)

<u>RELATED READINGS</u>:

Ghezzi Chapter 4 (pp. 61-115)
Pressman Chapter 10 (pp. 315-359)
Schach Chapter 10 (pp. 289-331)

Software Requirements Analysis

Input

Client request

Process

Identify customer needs

Output

Software requirements document

Software Specifications

Input

Software requirements documents

Process

Analyze and refine software requirements into testable specifications

Output

Software specifications document

Test plan/test procedures

Preliminary Design

Input

Software specifications document

Process

Generate a software architecture to satisfy the specifications

Output

Preliminary design document

Detailed Design

Input

Preliminary design document

Process

Refine each module in the preliminary design into detailed logic

Output

Detailed design representation

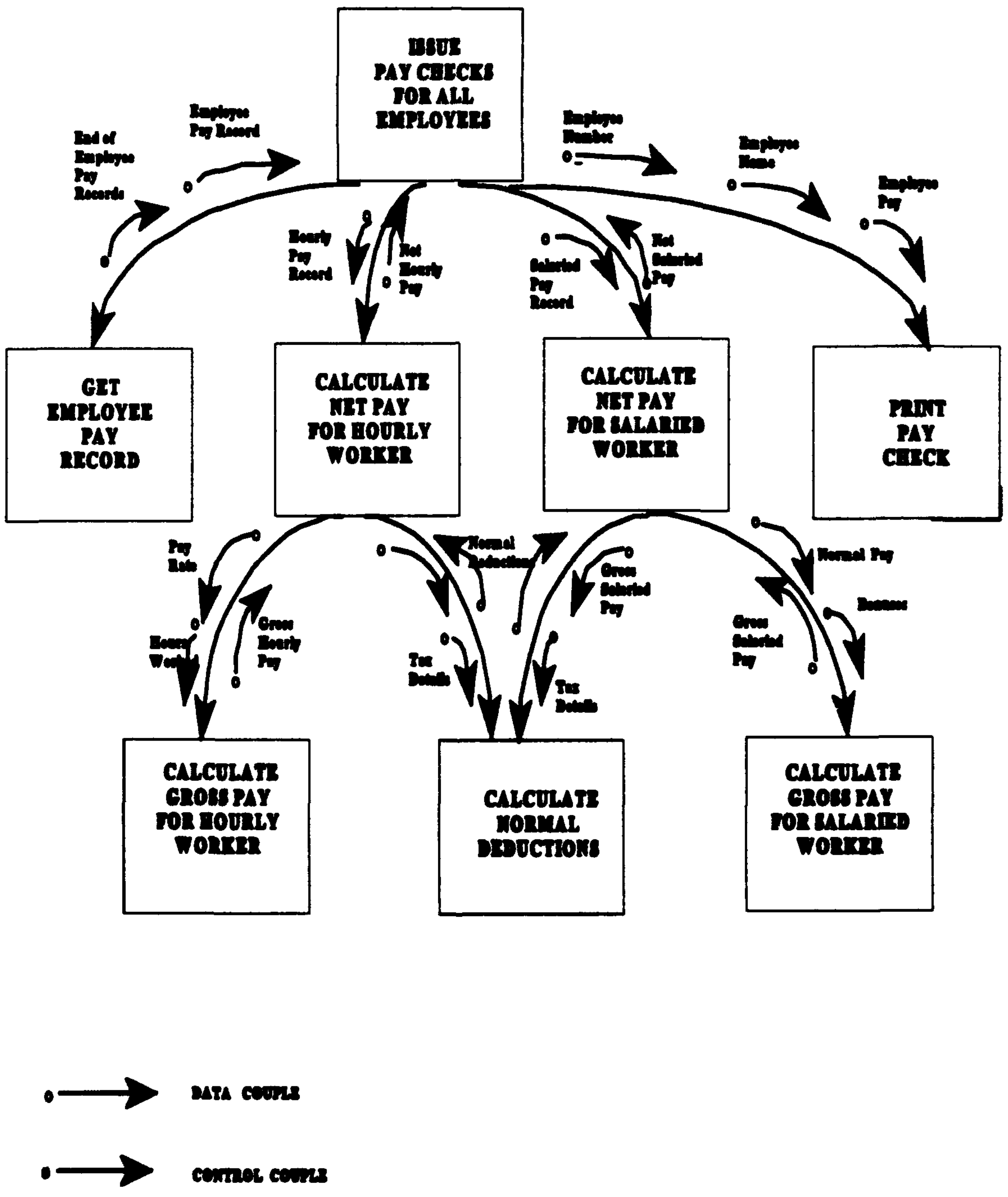
Key Design Goals

Maintainability

Reusability

Ease of testing

Example Structure Chart



LECTURE NUMBER: 007

TOPIC(S) FOR LECTURE:

Data flow diagrams and data dictionaries
Structure charts and data coupling
Requirements traceability

INSTRUCTIONAL OBJECTIVE(S):

1. Recognize and correct problems in data flow diagrams(DFDs).
2. Develop data dictionaries for DFDs.
3. Develop structure charts.
4. Understand relationship between test plans and requirements.

SET UP, WARM-UP:

(How involve learner: recall, review, relate)

(Write "WAYCY" on the board.) In some of your other programming classes you are given a program specification and are expected to immediately start coding. This method of software development has unfortunately become a software development methodology and has led to the acronym WAYCY -- Why aren't you coding yet?. As we have seen with the video rental system, a clear understanding of the system to be developed requires an iterative process. Recently we developed a DFD for the video rental system. Are we ready to code yet? (Display the context diagram for KoFF L5OH2.) There are still many questions that can be asked about the video rental system we are using as an example. Does this diagram represent the needed detail to build the video rental system? What is the information needed in "Membership application"? To determine that information we need to know the content of this data flow. This requires the development of a data dictionary and the development of an overall system architecture for the system.

(Learning Label- Today we are going to learn ...)

Today we are going to learn how the data dictionary is related to the analysis of requirements in a data flow diagram and how it is related to the development of a system structure chart, which is one possible representation of a system architecture.

CONTENTS:

1. Review the iterative nature of the software development process and discuss how changes made later in a system are more difficult to correct and more error prone.

   L5OH3
2. Display the first-level DFD for the video rental system. Ask them if the diagram is a complete and correct representation of the system.

L5HD2

a. Select requirements one at a time and trace it through the DFD. The completeness can be examined by checking the requirements list and the data dictionary.

L7HD1

b. Correctness can be evaluated only when the system is clearly specified. Hand out a complete data dictionary and examine several entries. Review the format adopted for this class for a data dictionary L4OH7, L4OH8

3. Another way to see if you clearly understand a system is to design a structure chart for it.

   a. A structure chart is not tied to a particular type of computer or programming language. It is a high-level design of the system showing the system architecture.

   b. There are many methods for deriving structure charts. One is to divide the system into its major tasks. Show the first level of the structure chart L7OH1. Be sure to specify that this is just one version of a structure chart.

   c. Discuss how "Enroll Members" and "Select Tapes" on the structure chart gather the major inputs to the system. Show how the major internal processing, including the dispensing and accepting of tapes, has been relegated to a single process.

4. The concentration of internal processing in one component (3.c above) can be used to introduce some issues about design including complexity and testing.

   a. Ask the students if having a single component doing all the internal processing is a good design. You are looking for them to be concerned about the complexity and the possibility for error.

   b. Discuss the concept of a central transform and how to reduce complexity. Redisplay the first-level DFD for the video rental system (L5OH3) and ask them where most of the information transformation takes place(Manage Tape Inventory). Discuss how the complexity might be reduced by separating out functions. Return to the discussion of the structure chart as a way to determine how to remove some complexity.

5. Work through the lower levels of the structure chart and review the concept of data coupling.

   a. Work through the "Enroll Members" structure chart. L7OH2 Use the data dictionary to determine what information needs to be passed to each component. Explain how this approach enables a clear division of labor. The goal of the "Enroll Members" component is to do one thing; to develop and pass the "new member letter" to the rest of the system.

   b. Do a high level examination of the "Select Tapes" component. Show overhead L7OH3 which just lists the sub-components of "Select Tapes". Use this to reinforce the concept of passing a data item to the rest of the system by tracing a members request for a tape into the lower levels of the structure chart.

   c. Carry the concept of passing information to the rest of the system to the discussion of "Manage Tape Inventory". Would the complexity of "Manage Tape Inventory" be reduced if it didn't really transform anything but was simply a switch for data going to and from other processes? Display the "Manage Tape Inventory" overhead L7OH4 and work through the Late Processing component.

6. Introduce the notion of testing at this stage of the life cycle.

   a. Test plans can be developed which are related to the requirements list. Test plans should contain details on specific tests to be conducted, tests can then be traced to certain requirements at any point in the software development life cycle. This is called requirements traceability. And requirements can then be traced to specific tests.

   b. Testing can be designed which is directly related to the structure chart. Even before any coding is started, tests which specify the interface requirements between system components can be specified and added to the test plan.

<u>PROCEDURE</u>:

<u>teaching method</u>:

Discuss the details of video rental DFD by asking questions which require the use of the data dictionary. Then present a method for building structure charts by dividing a system into inputs, processes, and outputs.

vocabulary introduced:

completeness
correctness
requirements traceability
data coupling
test plan

INSTRUCTIONAL MATERIALS:

overheads:

L5HD2	Revised Requirements List
L5OH4	Context diagram-video rental system
L5OH5	Level one DFD-video rental system
L7OH1	Top-level structure chart-video rental system
L7OH2	Structure chart for enroll members
L7OH3	Structure chart for select tapes
L7OH4	Structure chart for manage tape inventory

handouts:

L7HD1	video rental system data dictionary

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 005	CI 3, for small project is an exercise on this subject.

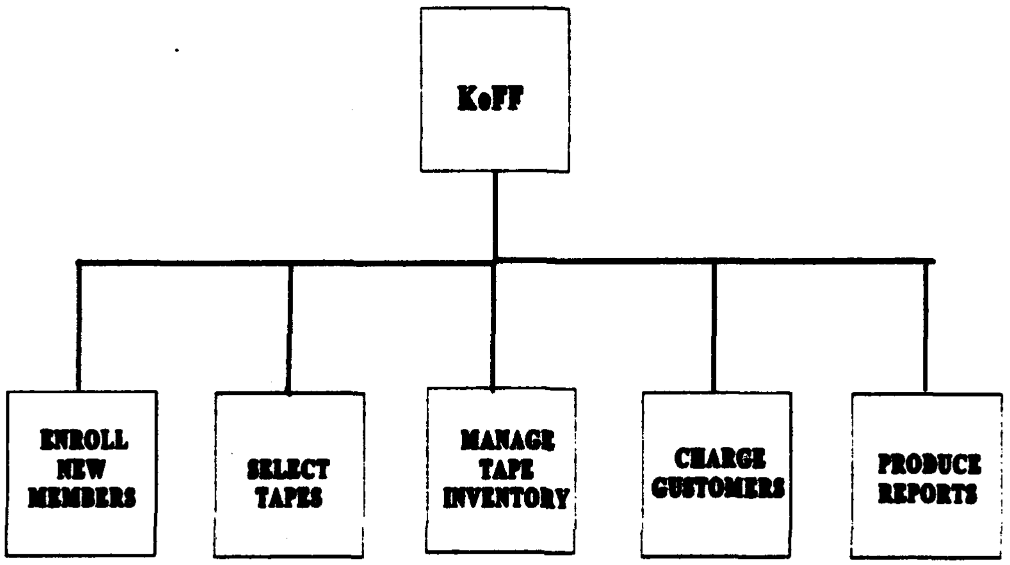
READING ASSIGNMENTS:

Sommerville Chapter 12 (pp.212-234)

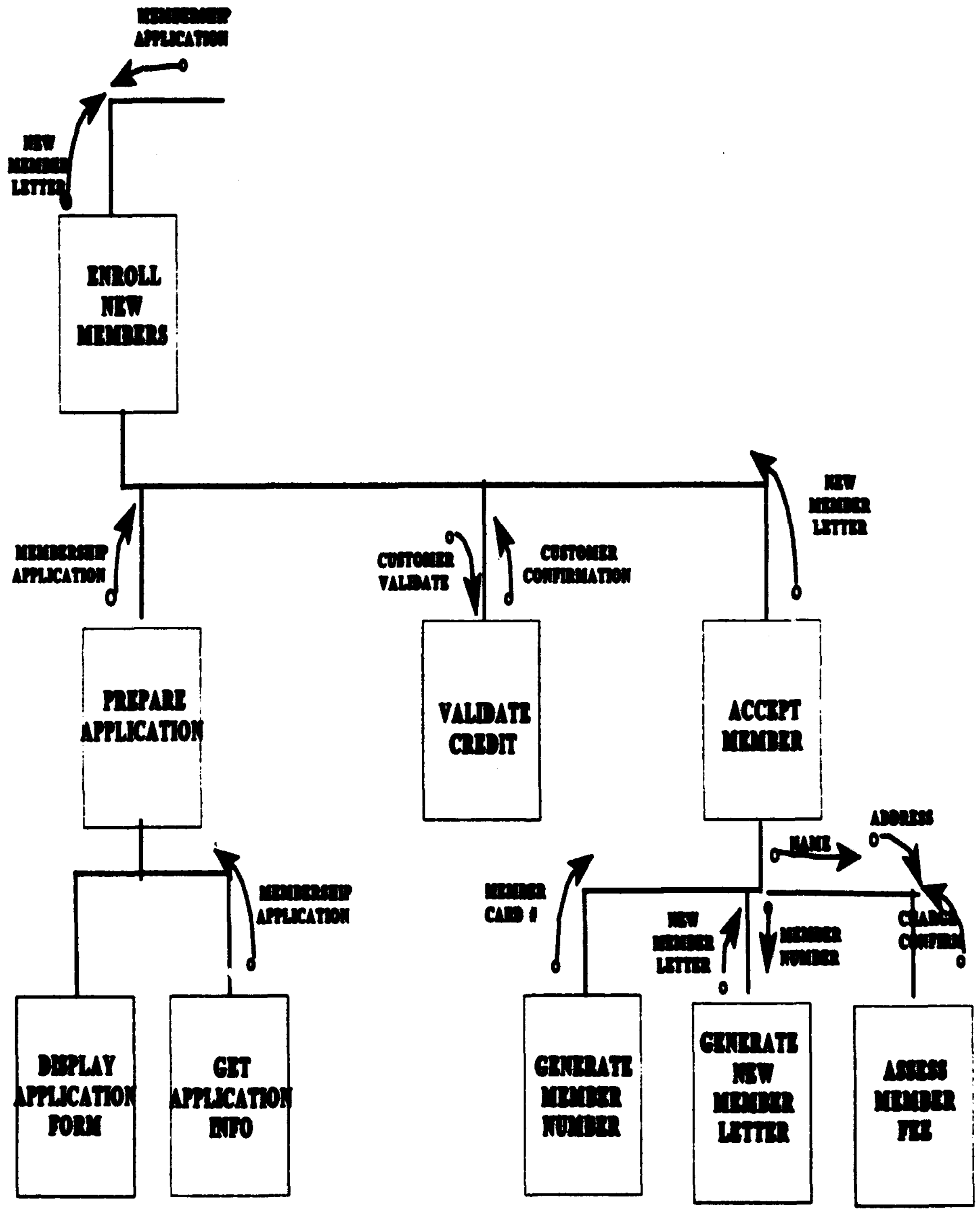
RELATED READINGS:

Berzins Chapter 3 (pp. 109-114)
Ghezzi Chapter 7 (pp. 394-400)
Pressman Chapter 11 (pp. 367-391)
Schach Chapter 10 (pp. 291-299)

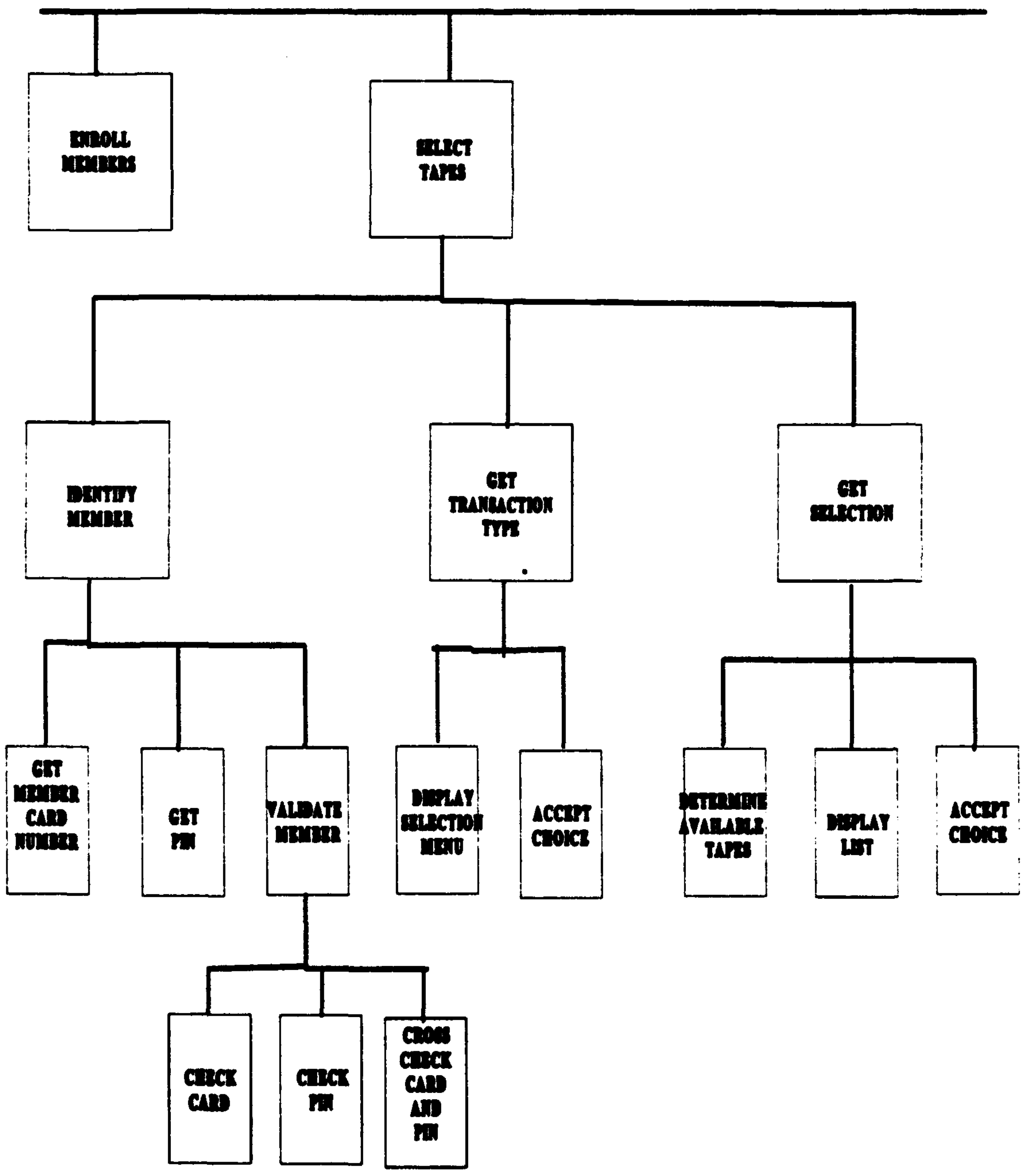
STRUCTURE CHART



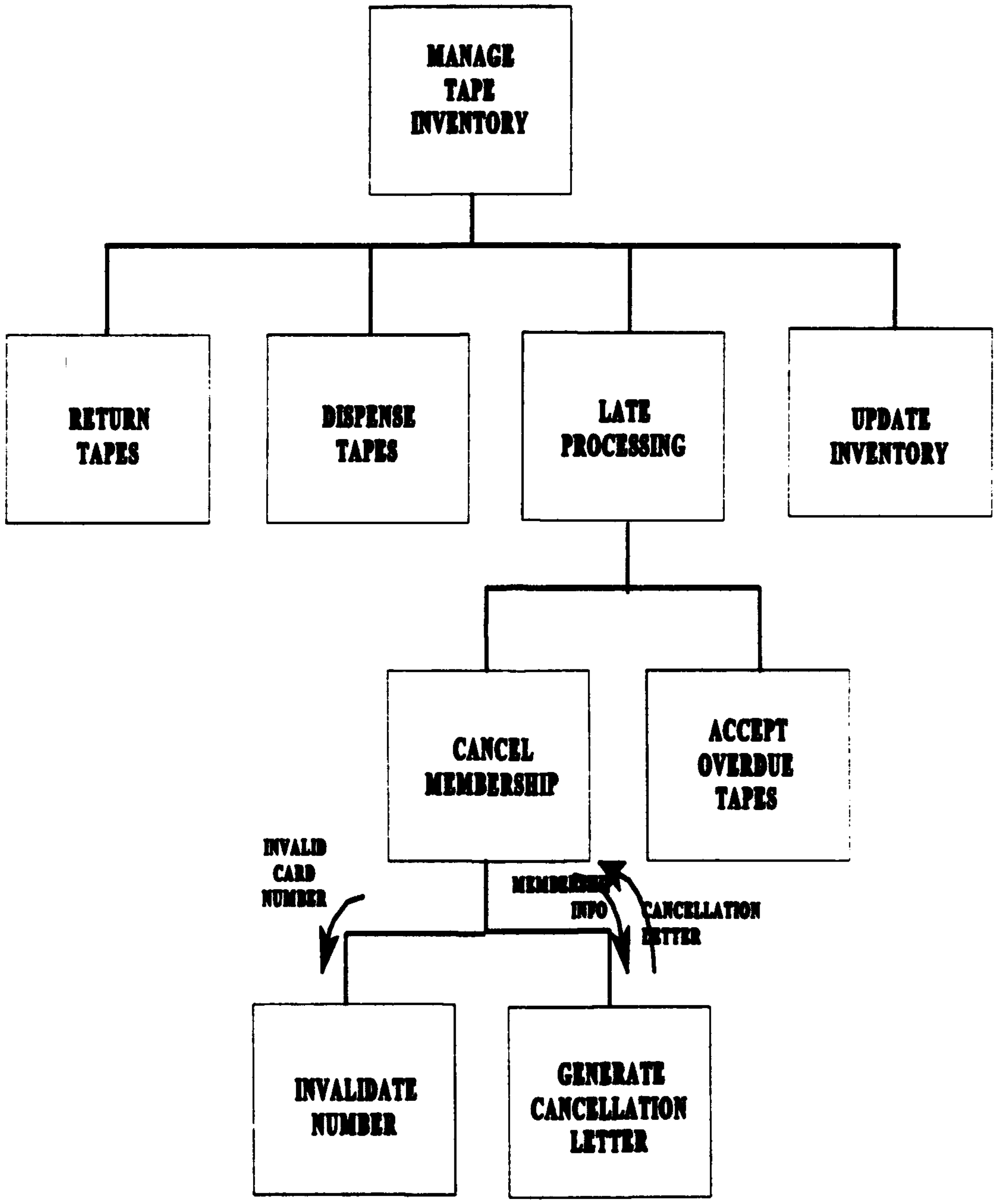
ENROLL NEW MEMBERS STRUCTURE CHART



SELECT TAPES STRUCTURE CHART



MANAGE TAPE INVENTORY STRUCTURE CHART



Data Dictionary for KoFF Example

access number = digit + digit + digit + digit

bar code = $\{ \text{digit} \}^{10}$

billing confirmation = not ok | ok + confirmation number

billing data = member card number + unique personal identification number + charge amount

blank = ' ' * blank character *

box number = $\{ \text{digit} \}^{8}_{1}$

cancellation letter = customer name + customer address + member card number

category = Adventure | Comedy | Children | Restricted

cents amount = digit + digit

charge amount = dollar amount + '.' + cents amount

charge card number = $\{ \text{digit} \}^{10}_{10}$

charge card # = charge card number

charge card type = American Express | Discover | Master Card | Visa

charge confirmation = not ok | ok + confirmation number

city = $\{ \text{letter} \}^{25}$

confirmation = ok | not ok

confirmation number = $\{ \text{digit} \}^{10}_{10}$

customer address = street + city + state code + zip code

customer confirmation = not ok | ok + confirmation number

customer name = $\{ \text{letter} \}^{25}$

customer validation = charge card type + charge card number + expiration date

day = digit + digit

dispensed membership card = * returned membership card *

dispensed tape = * returned tape *

digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

dollar amount = $\{digit\}^4_0$

duration = digit + digit

expiration date = month + '/' + day + '/' + year * last two digits only

invalid card # = member card number

letter = A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|blank

list of rental tapes = { movie info } * display selectable tapes to rent *

list of sales tapes = { movie info } * display selectable tapes to buy *

member card number = digit + digit + digit + digit + digit

member info = charge card type + charge card number + expiration date + customer name + customer address + telephone number + unique personal identification number + member card number + member status

member number = member card number + unique personal identification number

member status = valid | invalid

membership application = charge card type + charge card number + expiration date + customer name + customer address + telephone number

membership card = * physical card in system *

membership fee = charge amount

membership fee charge = charge card type + charge card number + expiration date + membership fee month = digit + digit

movie info = movie name + bar code + category + movie rating + quantity + transaction type + price

movie name = { letter }^{25}

movie rating = G | PG | PG-13 | R | NC-17 | X

movie request = movie name + transaction type + [duration]

new member letter = customer name + customer address + unique personal identification number + member card number + expiration date

prefix = digit + digit + digit

phone call = * call to customer about late tape *

price = charge amount

quantity = digit + digit

rental info = member card number + movie info

reports = * see KoFF description for this information *

returned tape = * tape brought back into KoFF *

sales info = member card number + movie info

state code = letter + letter

status change = member card number + invalid

street = street number + street name | 'P.O. Box' + box number

street name = { letter }$^{25}_{1}$

street number = { digit }$^{6}_{1}$

tape charges = charge card type + charge card number + expiration date + charge amount

tape selected = bar code

telephone number = prefix + access number transaction type = rental | sale

unique personal identification number = { digit $\}^{5}_{5}$

year = digit + digit

zip code = digit + digit + digit + digit + digit [+ digit + digit + digit + digit]

L7HD1

LECTURE NUMBER:008

TOPIC(S) FOR LECTURE:

General concepts of design
Architectural design
Behavioral design
Procedural design

INSTRUCTIONAL OBJECTIVE(S):

1. Recognize different classifications of design.
2. Understand different design stages.
3. Recognize different design techniques.

SET UP, WARM-UP:

(How to involve learner: recall, review, relate)

Once we have developed a complete set of requirements we have to transition from the question of what is wanted -- the solution space-- to an analysis and presentation of how we can achieve what the client wants. This process is called "Design" and it has several stages, just as requirements has several stages. Sommerville characterized design as "... a creative process which requires experience and some flair on the part of the designer."(page 176) Although there is some creativity required, the transition from the problem space to the solution space is a major step which requires some significant preparation and is accomplished in a series of stages.

(Learning Label- Today we are going to learn ...)

Today we are going to take a broad look at design and divide it into some manageable stages.

CONTENTS:

1. The design stage of the life-cycle.

    a. The product of design shows how a system can meet the user's needs as specified in the requirements. Whereas analysis defines the problem, the design shows how to solve it. The design is the basis for the implementation. Part of this design product is the user interface.

    L8OH1

    b. The goals of design have many similarities to the goals of requirements development: clarity, completeness, correctness, functionality, and continued usability. The design must be feasibility from both a technical and a practical perspective.

c. Design is a multi-staged process in the software development life cycle. The first stage of design - High Level or Preliminary Design - identifies the major component of a system and the relations between them. The second stage of design - Low Level or Detailed Design describes the internal characteristics of these components. Low level design also develops utility components of the system. Explain the ambiguity in "Design". It is both a process consisting of High level and low level design and a product of the life cycle which is used for implementation.

d. All implemented systems which have any longevity will have to change. A standard of well-designed software is its ease of maintainability. Parnas emphasized "Design for change!" as a mark of quality design.

e. When we design we should think of potential multiple applications of software. We add to Parnas' standard for quality "Design for reuse".

L8OH2

2. There are three distinct components of software design.

a. Architectural design - definition of the software structure; components and their relationships. This step requires a clear choice of how the system will be decomposed into components, e.g., modules in structured design, objects in object-oriented design. When decomposing into components, considerations include:

i design goals: modularity, low coupling
ii functional considerations: major system tasks
iii data storage activities
iv major system objects

Shows the environment and its interfaces, and constraints.

b. Behavioral design - a description of the way a system responds to specific inputs. This is a picture of the state a system will adopt given a description of the system's current state and its current input. A good example of state transitions is an elevator. If an elevator is on the floor above you and you press the up button. The elevator changes to the moving down state. If the elevator is on a floor below you, then pressing the same up button causes the elevator to change to the moving-up state. This shows how the same input can result in two different outputs.

c. Functional design - a system as a set of entities performing relevant tasks, and decomposed into relevant components. This view includes a description of the tasks performed by each entity and the interaction of the entity with other entities and with the environment. The architectural view is an elaboration of functional design, showing interfaces and information flows.

3. All three types are important for a complete design. The order will vary depending on the type of system being developed. These classifications provide a model for ways to partition the design process. For example the development of a telephone switching system might start with a behavioral design.

4. In an effort to achieve clarity of design, each design type has its own separate language or design notation which can be divided into a language-like notation and a graphical notation. L8OH3

   a. Architectural-structure charts, pseudo-code

   L8OH4

   b. Behavioral- Harrell state charts, control specifications

   c. Functional- data flow diagrams, process specifications, implementable components

   d. Ada as a textual notation can be used for all three components of software design.
      i can be abstractly stated
      ii not a large step to implementation

5. The process of design can also be divided into different stages

   a. Preliminary - the first step in a progressive transformation of the requirements
      i audience - customer, developer
      ii notation - graphical: structure charts, object diagrams
      text: prose descriptions

   b. Detailed - each of the components is designed in detail, algorithms and data structures are selected which are consistent with the interface established in preliminary design.
      i audience - design team, coders, technical staff
      ii notation - graphical: Nassi Shneidermann charts
      text: Formal specifications, pseudo-code, PDLs, Ada

PROCEDURE:

teaching method:

vocabulary introduced:

- architectural design
- detailed design
- behavioral design
- functional design
- Module Interface Language
- formal specifications
- preliminary design
- solution space
- user interface design
- Harrell State Transition diagrams
- control specifications
- process specifications

INSTRUCTIONAL MATERIALS:

overheads:

L8OH1	Goals of requirements
L8OH2	Distinct classifications of design
L8OH3	Examples of design notation
L8OH4	Example of a transition state diagram

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab006	Feedback on CI 2

READING ASSIGNMENTS:

Sommerville Chapter 10 (pp. 171-188)
Mynatt Chapter 4 (pp. 143-156)

RELATED READINGS:

Berzins Chapter 4 (pp. 207-216)
Booch Chapter 2 (pp. 35-38)
Booch(2) Chapter 2 (pp. 25-28)
Ghezzi Chapter 4 (pp. 61-115)
Pressman Chapter 10 (pp. 315-359)
Robert Firth, et al "A Classification Scheme for Software Development Methods," TR, SEI/CMU-87-TR-41, November 1987

COMMON GOALS OF REQUIREMENTS AND DESIGN

Clear

Complete

Correct

Functional

Testable

Maintainable

Reusable

Feasible

DISTINCT COMPONENTS OF DESIGN

a. Architectural - definition of the software structure

- i Design goals:
  Modularity, low coupling
- ii Functional considerations:
  Major system tasks
- iii Data storage activities
- iv Major system objects: object oriented

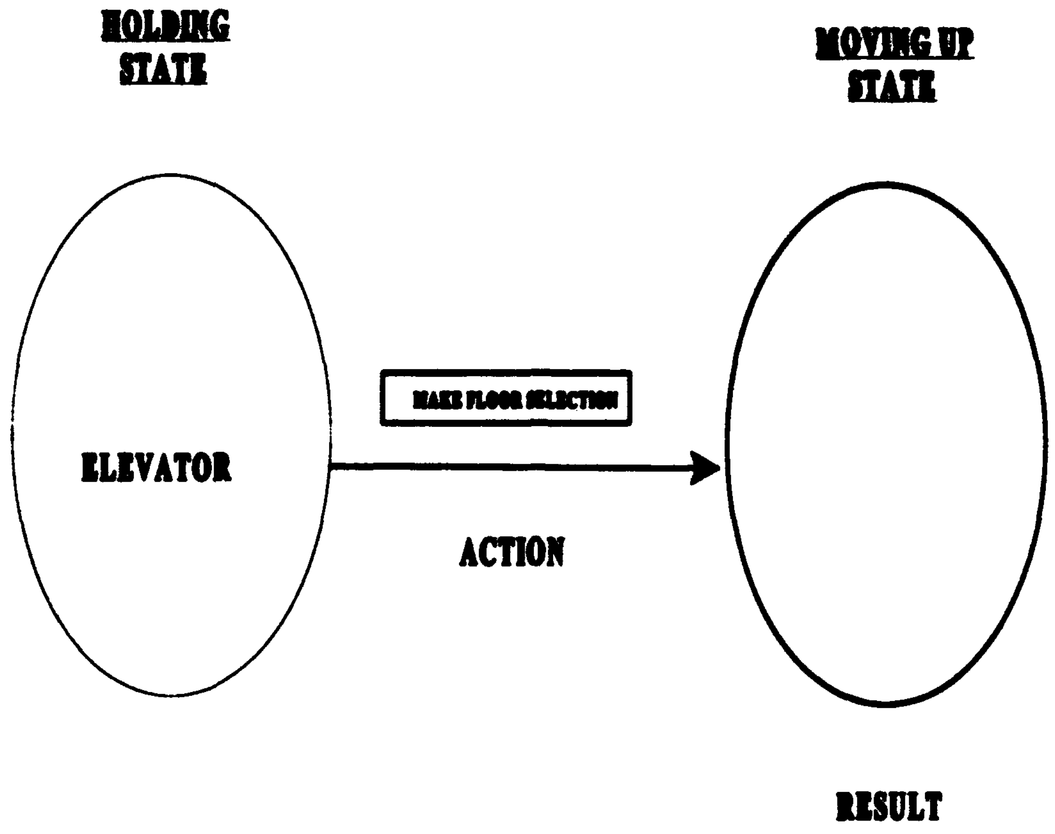
b. Behavioral - a systems as a set of transitions between states

c. Functional - a system as a set of entities performing relevant tasks

Examples of Design Notation

Classification	Graphic Notation	Text Notation
Architectural	Structure charts	Pseudo-Code
Behavioral	Harrell state charts	Control specifications
Functional	Data Flow Diagrams	Process specifications

EXAMPLE TRANSITION STATE DIAGRAM



LECTURE NUMBER:009

TOPIC(S) FOR LECTURE:

Testing
Test plans

INSTRUCTIONAL OBJECTIVE(S):
(indicate learner behavior expected or learning outcome)

1. Understand the different types of code testing.
2. Be able to develop a test plan.

SET UP, WARM-UP:
(How to involve learner: recall, review, relate)

We normally think of testing as only relevant to the coding aspects of a system and so we do not pay much attention to testing until the coding phase has already started.

As you will see later, testing is a process that can begin very early in the development life cycle and continues throughout the process.

This is a narrow view of testing. When we employ a broader view of testing we will develop a better product.

(Learning Label- Today we are going to learn ...)

Today, using the KoFF system, we are going to learn about the elements of a test plan and how the early development of a test plan improves requirements.

CONTENTS:

1. In the development of a system, we can divide the test into two basic categories --black box testing, and white box testing. Black box testing examines the external behavior of software, primarily testing that particular inputs result in their expected outputs. White box testing examines the internal structure and behavior of software. These two types of tests are employed at several testing stages of software development.

2. There are four hierarchically structured stages of testing, that most students are familiar with:
   a. unit: examines individual procedure as a stand alone components

   b. module: groups together units so that they can be tested together

   c. sub-system: groups together collections of modules and test both their internal correctness and their interface with other subsystems

d. systems: groups together sub-systems and test the entire system. This test for satisfaction of both functional and non-functional requirements.

There is a different type of testing, called acceptance testing, which occurs when a software product is delivered to a customer. The function of this type of testing is to prove that the software meets the needs stated in the requirements specification. Because this type of testing is tied to the requirements, a preliminary draft can be developed shortly after the requirements are finished. The specification of the precise system functions to be tested helps to clarify the requirements. One of the goals in the development of a preliminary test plan is to develop a requirements validation test matrix which related every requirement to a specific test or set of tests- called a test suite.

2. Test Plan contents
Using the attached instructors notes, work through the Preliminary test plan.
L9OH1

a. Preliminary Test Plan- There are many forms of test plans, but they all have similar goals in common, namely, to test that all requirements are satisfied and to record testing information in sufficient detail so that test can be repeated if necessary. To produce similar results it is necessary to record both the test that to be done and the order in which they are done, that is to generate a test schedule.

   In the development of a test plan, the test plan designer picks several categories of function, such as external access, to organize the test around.

L9OH2

b. The Test\Requirement Traceability Matrix connects each requirement from the requirements list to a particular type of test. Some requirements will have several tests associated with them such as requirement 2. Later in the design stage, other types of tests such as inspections and reviews will be included in the test plan. This matrix can also be used to specify , under demonstration, those tests which will be part of acceptance testing.

L9OH3

c. The test schedule is important because some test cannot be conducted until other stages of development have been successfully completed and tested. The test schedule provides information to the development team about the order in which they should produce their products.

L9OH4

d. The Status Report shows the progress of all testing up to the point of the report within the categories of testing : access, etc, decided upon by the test manager.

L9OH5

e. Test Results Form is used to keep track of the results of individual tests. Depending on the results of the tests and other information gathered during testing, the requires further analysis section may be filled out. Samples of how these forms are filled out are included as overheads. upon in the design of the test plan.

L9OH6

f. Tests to be Performed is the schedule of the order of the tes g within categories and the dependencies needed to be satisfied to preform any test.

L9OH7

g. Test Procedure Form spells out the low level structure of the tests to be preformed within a suite of tests.

3. Building a test plan provides you with a way to validate requirements.
   a. Example - in the KoFF system, by designing access tests it was discovered that a way was not described in the system requirements for the owner to obtain legal access to the system.
   b. The plan also helps avoid incomplete testing. Failure to complete all stages of testing can have significant consequences. Example of incomplete testing - Hubbell telescope - all the pieces of the system were tested but the pieces were not integrated and tested together. This resulted in an expensive system failure.

<u>PROCEDURE</u>:

<u>teaching method</u> :

(types of activities)

This lecture contains a handout -- KoFF Video Rental System Preliminary Test Plan-- and a set of instructor notes explaining the plan. The students need to have a copy of the plan in their hands to follow the lecture and to be able to use it as a model for the development of their own plans.

<u>vocabulary introduced</u>:

unit testing
module testing
subsystem testing
systems testing
test plan
system failure

integration testing
requirements validation matrix

INSTRUCTIONAL MATERIALS:

overheads:	
L9OH1	Preliminary test plan (KoFF system)
L9OH2	Test requirement\traceability matrix
L9OH3	Test schedule
L9OH4	Status report
L9OH5	Test results form
L9OH6	Tests to be performed
L9OH7	Test procedure form
handouts:	KoFF test plan

RELATED LEARNING ACTIVITIES:
(labs and in class exercises)

Lab007- Have student develop classes of test which would include all of the requirements for their projects

READING ASSIGNMENTS:
Sommerville Chapter 19, 22 (pp. 378-88, 425-441)
Mynatt Chapter 7 (pp. 276-315)

RELATED READINGS:
Ghezzi Chapter 6 (pp. 260-297)
Pressman Chapter 13 (pp. 631-659)

KoFF Video Rental System
PRELIMINARY TEST PLAN[1]

1.0 Introduction

The test plan is presented in sections 3.0 and 4.0 of this document. Section 3.0 describes the methodology to be used for the testing process. Section 4.0 contains test procedures to be executed. These procedures are derived from the requirements specification document for the KoFF Video Rental System. Test data developed will be included in the Appendix. The test results will also be included in the Appendix at the completion of the testing.

2.0 Referenced Documents

Electronic Fund Transfer and Charging Standards....
KoFF Client Request dated June 11, 1993.
KoFF Data Flow Diagram dated June 14, 1993.
KoFF Preliminary Design documents dated June 15,1993.

3.0 Test Methodology

The following paragraphs will describe the items to be considered in the planning of the tests for the KoFF Video Rental System.

3.1 Test Group Involvement

Considering the size of the test group and the short time period allocated for the test activity, the test group will participate in the subsystem test, perform the integration test and then demonstrate the acceptance test. The participation in subsystem testing is limited to observing the designer's test so that the test group is familiar with the use of the system.

3.2 Requirements Traceability

The methodology for showing traceability of the requirements to the test is be that the requirements are identified by line number in the requirements list, rather than by paragraph number in the client request. The method for verifying the requirements is identified. The methods used are: 1) inspection of code, hardware , or execution results; 2) test and analysis of test results; and 3) demonstration of the system. Similar requirements will be grouped in the test procedures.[2] The test/requirements traceability matrix is figure 3.2-1.

[1] This plan presumes a test team of four experienced software engineers.

[2] Another technique is to group requirements by major system functions.

Test/Requirement Traceability Matrix

Requirement	Test/Test Method		
	Inspection	Test/Analysis	Demonstration
1		D4	
2(1)		A2	
2(2)		D4	
2(3)		D7	
2(4)		D10	
3		D8	
4		C11	
5		C7	
6		B1	
7		A2	
8		C3	C3
9		C4	C4
10		B7	
11(1)		C1	C1
11(2)		C6	
12(1)		A2	
12(2)		C1	
12(3)		C3	
12(4)		C8	
12(5)		C12	
13			
14		C13	C13
15		D9	D9
16		C2	C2
17		D14	
18(1)		C5	C5
18(2)		C6	
19(1)		A1	
19(2)		A2	
19(3)		C1	
20		B2	
21		C15	C1
22		C16	

Figure 3.2-1

3.3 TEST SCHEDULE

The planning schedule for the tests is in Figure 3.3-1. The schedule identifies the plan for completing the plan, developing procedures, test data sets, test software, and test execution.

Test Schedule

Start Date	Complete Date	Activity
June 10	June 15	Complete Test Plan
June 28	June 30	Order test equipment
June 30	July 12	Develop test procedures
July 12	July 15	Generate Test Data
		<u>Test to be performed</u>
July 16	July 18	ACCESS TO THE SYSTEM integration order 1-5
July 19	July 23	SELL TAPES integration order 6-9
July 24	July 27	RENT TAPES integration order
July 27	July 27	REGULAR CHARGES integration order
July 27	August 5	LATE CHARGES integration order
August 5	August 7	MANAGING TAPES integration order
August 7	August 10	REPORTS integration order

Figure 3.3-1

3.4 Status Report and Problem Report

A form showing the means for tracking and reporting the testing of the system is included in Figure 3.4-1. A problem reporting form was not used.

Function	Number's of Test Procedures	Procedures Scheduled/Executed/Successful	% Success
A. Access			
B. Inquiry/Selection			
C. External Responses			
D. Add/Delete/Update			

Figure 3.4-1

3.5 Test Procedures/ Results

Figure 3.5-1 will be used to describe test procedures. Figure 3.5-2 will be used to describe the process for recording test results, including version tested, date tested, and results will be described. The forms to be used are in Figure 3.5-1 and 3.5-2.

TEST PROCEDURE FORM

Function:

Procedure:

Requirements:

Prerequisites:

Test Data Required:

Test Steps:

Analysis Required:

Figure 3.5-1

Test Results Form

Test Procedure:

Date Test Executed:

Version Number Tested:

Test Results:

  Problems Identified:

  Analysis Results:

Retests Required:

Figure 3.5-2

The test procedures are developed from the requirements and user documentation for integration testing.

The first procedures will be for a test of the integrated system functionality. It will quickly answer the question, "Is it worth proceeding to perform the detail tests?"

Then the detailed procedures for each requirement, or group of requirements, will be developed. These procedures will take into consideration testing the limits as well as the normal input data cases. Criteria for evaluation of results will be included. The input data needed will be defined. Any processes that need to be developed to prepare or evaluate the data will be de'ined.

On most projects, the integration and acceptance tests would not be included on the same test plan. However, it appears that would be appropriate on this project. The acceptance tests will also be developed from the requirements document and also use to the maximum extent possible the actual data available. These tests will have the objective of testing the use of the system in the environment of the user community.

4.0 Test Procedures

Function/Procedure		Tests to be Performed Order of Tests	Successful Prerequisite
Integration Tests			
A.	Access		
1.	Customer legal card	3	D.4
2.	Customer card expired	4	D.4
3.	Customer card invalid		D.4, D9
4.	Owner legal access		
B.	Inquiry/Selection		
1.	Rental Inquiry	10	A.1, D.4, D.5
2.	Sales Inquiry	5	A.1, D.4, D.10
3.	Tape Rental Report		
4.	Sales Report		
5.	Customer Rental Report		
6.	Tape selection	6, 11	
7.	Duration selection	12	
C.	External Responses		
1.	Accept/Return Member Card	2	
2.	Capture invalid card		
3.	Capture Expired Card		
4.	Rental Dispensing		
5.	Sales Dispensing	8	
6.	Tape Charges	7, 13	
7.	Membership charges		
8.	Late Charges		
9.	Restocking Charges		
10.	Validate charge card		
11.	Send new member information		
12.	Accept input tapes	14	
13.	Late notice phone call		
14.	Send member removal information		
15.	Send membership information		
16.	Send rental tracking information		
D.	Modify/Update/Add		
1.	Change video tape titles		
2.	Change video tape to sale item		
3.	Change Video tape prices		
4.	Add new member	1	
5.	Change movie rental information		
6.	Change customer status to invalid		
7.	Create membership number and pin		
8.	Add charge card information		
9.	Invalidate membership card number		
10.	Change Sales inventory information	9	

Figure 3.5-1

TEST PROCEDURE FORM

Function:	A.1
Procedure:	Customer legal access
Requirements:	1
Prerequisites:	Legal membership
Test Data Required:	Valid membership card number

Test Steps:

a. Insert valid card into system
b. Verify rental sales option screen displayed
c. Select quit
d. Verify card inserted is card returned
e. Insert a card which is not an RRR membership card *
f. Verify display of "Not an RRR card"
g. Verify card inserted is card returned

Analysis Required:
(note: The requirements did not specify what to do if a non-RRR card was inserted. The test designer made a design decision here.)

L9OH7

TEST PROCEDURE FORM

Function: A.2

Procedure:	Process expired card
Requirements:	8
Prerequisites:	Legal membership
Test Data Required:	Valid membership card number with expired date

Test Steps:

a. Insert valid, but expired card into system
b. Verify expired card screen is displayed
c. Verify card is captured
d. Verify membership file is updated *
e. Verify that the "Welcome to KoFF" screen is displayed after 90 seconds.

Analysis Required:

Check timing for screens

(note: Requirements did not say how to track expired cards.)

L9OH7

TEST PROCEDURE FORM

Function:	A.3
Procedure:	Customer legal access
Requirements:	16
Prerequisites:	Legal membership, invalidated card
Test Data Required:	Invalid membership card number

Test Steps:
- a. Insert invalid card into system
- b. Verify invalid card screen is displayed
- c. Verify card is captured
- d. Verify membership file is updated *
- e. Verify that the "Welcome to KoFF" screen is displayed after 90 seconds.

Analysis Required:
(note: Requirements did not say how to track expired cards.)

L9OH7

TEST PROCEDURE FORM

Function:	B.1
Procedure:	Process Rental Inquiry
Requirements:	6
Prerequisites:	Legal membership
Test Data Required:	Valid membership card number, list of available rental tapes

Test Steps:

a. Insert valid card
b. Verify that rental/sales selection screen is displayed
c. Select Rentals
d. Verify that rental selection screen is displayed
e. Verify that all and only available tapes are displayed
f. Select quit
g. verify that "Thank You screen is displayed"
h. Insert valid card
i. Verify that rental/sales selection screen is displayed

L9OH7

j. Select Rentals
k. Verify that rental selection screen is displayed
l. Verify that all and only available tapes are displayed
m. Select tape
n. Select duration
o. Verify that customer's account is charged
p. Verify that the correct tape is dispensed
q. Verify that available tape list is changed
r. Verify that membership history is changed
s. Verify that "Thank You" screen is displayed for 30 seconds
t. Verify that the "Welcome to KoFF" screen is displayed.

Analysis Required:

L9OH7

TEST PROCEDURE FORM

Function:	B.1
Procedure:	Process Rental Inquiry (no duration selected)
Requirements:	6
Prerequisites:	Legal membership
Test Data Required:	Valid membership card number, list of available rental tapes

Test Steps:

a. Insert valid card
b. Verify that rental/sales selection screen is displayed
c. Select Rentals
d. Verify that rental selection screen is displayed
e. Verify that all and only available tapes are displayed
f. Select tape
g. Verify that within 45 seconds "please select duration is displayed"
h. Select duration
i. Verify that customer's account is charged

L9OH7

j. Verify that the correct tape is dispensed
k. Verify that available tape list is changed
l. Verify that membership history is changed
m. Verify that "Thank You" screen is displayed for 30 seconds
n. Verify that the "Welcome to KoFF" screen is displayed.

Analysis Required:

L9OH7

Preliminary Test Plan Instructor Notes

Numerous test plan models exit. This test plan model is designed to show students how to trace tests to specific requirements. It also can be used to show students how the development of a test plan can act as a verification technique for requirements.

The plan is divided into four major sections. The introduction outlines the structure of the plan and how it relates to a particular system. The second section on referenced documents should mention those system development documents which were used to build the plan. It should also include reference to documents which specify special constraints for the system. Because the video rental system automatically charges the customer's accounts, its processing must conform to the electronic funds transfer act. Knowledge of these standards is needed to construct adequate tests of the customer charge card functions. The third section specifies the test methodology and the fourth section lists specific test procedures. In this plan they are in the form of test scenarios. Other techniques would include specific code or the results of automatic test generators. They would also include justification for the choice of particular test cases as effective test cases. This is a preliminary plan built at an early stage of the life cycle so low level code details are not included. At this stage of development, the presumption is that detail testing will be completed separately and the primary function of this plan is to specify integration testing.

The order of the major sections in the report does not reflect the order in which major decisions are made about the test plan. Section 3 is the major body of the plan. Section 3.2 is the traceability matrix which is used to trace failed tests to particular requirements. The requirements numbers refer to the KoFF Adjusted Requirements List. The specific tests or test methods for each requirement are filled in from the test procedures listed in section 4.0. Section 3.3, the test schedule, is also dependent on section 4.0.

The first step in the development of this test plan is to divide the system into its major functions. The selection of these functions will determine all the other elements in the plan. In the development of this plan, the major function we selected were: system access, inquiry and selection, external responses, modify the data-update, add, delete. These are listed under major functions in the Test Status Report (Figure 3.4-1). The remainder of that report is filled out as the system is developed. These are the categories used to group integration tests.

The second step is to subdivide these integration test categories and fill the tests to be performed (Figure 3.5-1) in section 4.0. The integration tests are listed as sub-functions. For example, the sub-functions under access include attempts to access the system with all status of membership card and attempts to access the system by Mr. Richard. The subfunctions are determined by referring back to the requirements. A general testing strategy is determined, which is listed in the test schedule (Figure 3.3-1). In this example it is: test access to the system, test selling tapes, test renting tapes, test regular charges, test late charges, test membership management, test tape management, and test writing reports. This is used to determine the order of testing. Before anyone can access the system they must be a member so the first function to test is D.4. Then A.1 customer legal access can be tested. The sequence of the test for access are listed. Many integration tests get repeated as major elements in the testing strategy are visited. The tape selection B.6 tests are executed in the rental tapes test and in the sales tapes tests. This is a good way to introduce a discussion of regression testing. After the order of integration testing is specified, the specific integration test which must be complete in order to begin the current integration test is listed under "successful prerequisite".

At least one problem with the requirements surfaces when students try to list the successful prerequisites for integration test A.4 (Owner legal access) in section 4.0. The requirements are quite vague about the way Mr. Richard will access the system. If it is unknown whether he wants to do his updates at the Kiosk by using a special access card or do his updates remotely, then there is no way to test this requirement. Working through a preliminary test plan shows an unstated requirement.

The third step is to use the table developed in section 4 under Test to be performed and return to section 3 to fill in the test matrix. For example, the first requirement in the revised requirement list is to accept membership. During system integration this is tested by test set D.4. In the requirements matrix list D.4 in the same row with requirement 1. Follow this procedure when filling out the rest of the matrix.

The last element is to give complete test procedure or scenarios using Test Procedure Forms. A complete test plan will have a test procedure form tied by function name to each integration test. The scenario for "Legal Access" will be tied to integration test A.1. The scenarios represent the external behavior of the system and can be specified at this stage of development. They are also useful requirements clarification tools because they are they can be understood without a technical background and they sometimes reveal missing or incomplete requirements (see notes to Test Procedure Forms).

The procedures related to particular functions are filled-in during detailed design. The test plan again functions as a test of a higher life cycle stage. When the test are actually performed, the test results form is filled out and attached to the test procedure form. When all testing is complete, the test status form is completed.

LECTURE NUMBER: 010

TOPIC(S) FOR LECTURE:
Ada as a Design Notation

INSTRUCTIONAL OBJECTIVE(S):

1. Make students aware of another approach to designing a solution, i.e., an object-oriented approach.
2. Introduce Ada package specifications as part of a design notation.

SET UP, WARM-UP:
(How to involve learner: recall, review, relate)
(Learning Label- Today we are going to learn ...)

In a previous lecture, the concept of design was introduced, using a functional approach. This lecture looks at a more object-oriented approach to design where the system is decomposed into subsystems and the associated information and actions/tasks are delineated.

CONTENTS:

L10OH1

1. Discuss criteria for "good software design"

   a. A design should be readily understandable.

   b. A design should be readily modifiable.

   c. A design should be testable.

   d. A design should be reusable.

2. Ada as a design tool

   L10OH2

   a. Ada notation allows us to perform the necessary design tasks of high-level or architectural design, interface design, component specification, algorithmic or low-level design.

   L10OH3

   b. Three classes of subsystems can be used for this type of decomposition: user utilities e.g., directly involving the user; resource management utilities managing the data stores; and service utilities. Design can be begun by decomposing the system into subsystems or design objects.

L10OH4

c. Working from the analysis documents (the requirements list, context diagram, data flow diagrams, and data dictionary) one can make a preliminary list of the major subsystems. For example, consider the dfd for the KoFF video rental system. The following subsystems are derived from the KoFF subsystem L5OH3:

    member
    tape
    rental
    charge card
    reports
    screens

The first draft of the Ada specification for the Member subsystem was presented to show how Ada provides a design notation that is understandable, modifiable, and testable.

d. Consider the member subsystem. Identify the major processes or actions, data and attributes required for this system. L10OH4 Using this list one can develop an Ada package specification for the members of the subsystem. The details of the member subsystem are hidden in the procedures and functions.

e. Develop similar package specifications for L10OH4, L10OH5,OH6,OH7,OH8,OH9
This Ada specification is not a complete one.

PROCEDURE:

teaching method and media:

vocabulary introduced:
subsystems
user utilities
resource management utilities
service utilities
Ada specifications

INSTRUCTIONAL MATERIALS:

overheads:

L10OH1	Software Design
L10OH2	Ada as a Design Tool - design tasks
L10OH3	Ada as a Design Tool - subsystems
L5OH4	KoFF system - context diagram
L5OH5	KoFF system - level 0 diagram

L10OH4	KoFF Subsystems - Member
L10OH5	KoFF Subsystems as ADA specifications
L10OH6	KoFF Subsystems - Tape
L10OH7	KoFF Subsystems - Rental
L10OH8	KoFF Subsystems - Charge Card
L10OH9	KoFF Subsystems - Reports, Screens

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)
Lab 008 - Design review team presentations for small projects

READING ASSIGNMENTS:
Sommerville Appendix A pg 607-620

RELATED READINGS:
Booch Chapter 4 (pp. 28-43)
Booch(2) Chapter 2 (pp. 17-32)

Software Design

Criteria for "good software design"

- Design should be readily understandable
- Design should be readily modifiable
- Design should be testable
- Design should be reusable

Ada as a Design Tool

Design tasks

- High-level or architectural design
- Interface design
- Component specification
- Algorithmic or low-level design

Ada as a Design Tool

Three classes of subsystems

User utilities
items specified in the requirements

Resource management utilities
responsible for some resource within the system

Service utilities
provides services to other subsystems

KoFF Subsystems

Member

information:
- name
- address
- telephone number
- charge card info
  - type
  - number
  - expiration date
- member card number
- unique personal identification number
- member status

actions on this information:
- enroll a member
- invalidate a member
- modify a member's information
- generate membership numbers
- check membership status
- retrieve charge card number

KoFF Subsystems as Ada Specifications

```
package MemberPkg is

   type Member is private;

   procedure Enroll (a_member : Member);

   procedure Invalidate
                        (a_member : Member);

   procedure Modify (a_member : in out
                         Member);

   procedure Generate (for : Member;
                        card_number : out integer;
                        id_number : out integer);

   procedure Retrieve (Charge_card_number :
                              out integer;
                         for_a : Member);

   function Check_membership_status (for_a :
                      Member) return Status_type;

private
   type Member is ...
end MemberPkg;
```

KoFF Subsystems

Tape

information:
- movie name
- category
- movie rating
- quantity
- transaction type
- price
- info per tape
    - bar code number
    - availability status

actions on this information:
- buy a tape
- update inventory
    - add tape
    - delete tape
    - modify tape title
    - modify tape quantity
    - modify tape transaction type
    - modify tape price
- change availability status
- determine available rental tapes
- determine available sales tapes

KoFF Subsystems

Rental
(associates MEMBER and TAPE)

information:
- member card number
- bar code number
- check out date
- duration length

actions on this information:
- rent a tape
- return a tape
- process 5-day late tape
- process 10-day late tape

KoFF Subsystems

Charge Card
- confirm customer charge card
- bill membership fee
- bill rental fee
- bill sales fee
- bill 10-day late fees

KoFF Subsystems

Reports
- generate new customer information report
- generate membership cancellation letter
- generate tape rental report
- generate customer rental report
- generate detailed financial report
- generate summary financial report

Screens
- display list of available rental tapes
- display list of available sales tapes
- display selection menu
- display membership application info
- get application information
- get rental information
- get sales information

LECTURE NUMBER: 011

TOPIC(S) FOR LECTURE:
Software maintenance

INSTRUCTIONAL OBJECTIVE(S):

1. Know the relative effort and costs associated with software maintenance.
2. Distinguish between the three types of maintenance (corrective, adaptive, perfective).
3. View performance of maintenance as a rerun of the software development process, involving analysis, design, implementation, testing, and associated documentation.
4. Understand preventive maintenance and its importance.
5. Appreciate the need to effectively manage and control change.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

Refer back to early lectures on activities involved in software development. Use overhead L11OH1 to review phases. This chart depicts the percent of effort for the various phases in developing a software product. Today we're going to talk about what happens after the product has been delivered and is being used.

(Learning Label- Today we are going to learn ...)

CONTENTS:

1. Discussion - What is maintenance, in general? In the context of hardware (e.g. automobiles) this generally means fixing a mistake. What is maintenance in the context of software? In the context of software, maintenance is much broader and includes enhancements and additional functionality as well as fixing mistakes. [Note: Pressman's discussion of the differences between hardware and software is particularly useful here. See item 5 below.] Managing maintenance and controlling maintenance costs are significant issues in software engineering. Software maintenance encompasses all of the activities relevant to the software product that occur after it has been delivered and installed.

2. Discussion - What are these activities; i.e. what are the different types of maintenance?

   a. Corrective - fixing defects that were not discovered during system development.

Examples - An auto manufacturer recalling vehicles; a pacemaker manufacturer discovering a software defect after installation.

b. Perfective, or enhancement - adding new features and functionality to system. Note that Sommerville disagrees with this definition when he says perfective maintenance improves system <u>without</u> changing its functionality.

Example - adding a Thesaurus to a word processor.

c. Adaptive - modification in order to respond to changes in the environment in which system is operating.

Examples - change in income tax package every year due to changes in tax laws; adding mouse capability to a program.

d. Characterize perfective and adaptive as "good" maintenance in the sense that this activity is not an indicator that the software is bad; perhaps is an indicator of success, that it is being used and is capable of being modified. Characterize corrective as "bad" maintenance in the sense that it does indicate a defective product.

3. Maintenance costs/effort.

a. Relative to amount of effort spent on development (up to time product is delivered and installed), how much effort is devoted to product maintenance (everything after delivery and installation)?

L11OH2
Note that more than twice as much effort is expended to maintain product as to develop it. Therefore anything that can lead to reducing maintenance costs is a significant contribution.

This should make it clear why maintainability is consistently listed as a key goal in software development (recall it is one of Sommerville's key characteristics of well-engineered software; it is a key consideration during design; writing maintainable code has been drilled into you in programming courses).

b. L11OH3 reports on experiences of several large companies on the relative costs of fixing a defect at various stages in the development process. For example if it takes $10 to detect and correct a fault during the implementation phase, that same fault could have been corrected for only $2 if caught during specification. Similarly, it will cost $200 if not detected until after the system is delivered and in use. The key point, in either case, is that the earlier errors are detected and corrected, the less cost and effort is involved.

c. L11OH4 is based on industry data where identified defects were classified as requirement definition problems, design problems, etc. Note that half are requirements problems, and 75% are pre-implementation errors.

   MORAL - It pays to find faults early.

d. Discussion question - Which of the types of maintenance (corrective, perfective, adaptive) do you think is most prevalent? least prevalent?

   L11OH5. Point out and discuss the misperception that most maintenance is corrective.

4. Approach to maintenance

   a. Consider a maintenance scenario. A software <u>product</u> is developed using a sound software engineering <u>process</u>, is installed, and is being used successfully by satisfied customers. Based on use, it becomes apparent that a new capability would make things even nicer. The customer requests the new feature (perfective/enhancement maintenance request). What does the software organization do? Discuss each of the major software life cycle phases and let the discussion bring out that maintenance is actually a rerun of the software development process (requirements definition, specification, design, implementation, testing). L11OH6

   b. What qualities should the personnel involved in maintenance possess? Let discussion bring out that they need analysis skills, design skills, etc.

      Unfortunately attitudes towards maintenance tasks are often inconsistent with the importance of maintenance. It seems to be undervalued; some organizations may talk about the importance of maintenance but not put their money where their mouth is. Some manifestations this mistake include the following:

- i The role of maintenance programmer is an entry-level position in some organizations. You put the new kid there and their first promotion is out of maintenance.
- ii Maintenance is not sufficiently emphasized as an important criteria for acceptance of the product.
- iii Mention specific suggestions by Boehm in Sommerville.

c. Introduce preventive maintenance by asking if they can think of another type of maintenance term, outside software, that they've heard. Extract the term preventive maintenance. What does the term mean to you in general; what sorts of things does it entail? While not specifically addressed in most textbooks, Pressman suggests that preventive maintenance is a fourth type of maintenance activity. It occurs when software is changed to improve future maintainability or reliability, or to provide a better basis for future enhancements.

Point out that it is normally not difficult for software developers to anticipate/predict requests for change that are likely to be made after the product has been in use for some period of time. Discuss this, give examples, and relate this to the students current projects. Stress that this knowledge of likely areas of change can be used by designers and implementors in assuring that the software is amenable to change; i.e. is maintainable.

5. The following discussion of the differences between hardware and software is optional but, if time permits, is an excellent introduction to the topic of software maintenance. The discussion is from Pressman, pages 10-13.

Begin by asking "what are the differences between hardware and software". Present the following differences given by Pressman.

a. Software is developed/engineered; not manufactured. E.g. developing hardware involves analysis, design, etc but it is ultimately manufactured. What is the manufacturing phase for software? There is essentially no such phase for software. Consider automobile; when are defects introduced into a car? Many occur during manufacturing phase (e.g. "Monday cars"). We should have little of that with software since it's a simple replication process. The key point here is that hardware costs are concentrated in manufacturing and that is not true for software. Therefore a whole category of defects that are frequent in hardware should not even exist in software.

b. Software doesn't wear out. Sketch the failure rate curve, Pressman Figure 1.2, for hardware. Discuss:

i <u>Why high failure rate early?</u> hardware exhibits high failure rates early (WHY? - design/mfg defects)

ii <u>Why drop?</u> Defects corrected & failure rate drops to steady state for some period of time

iii <u>Why rise again?</u> After a while failure rate rises as hardware suffers from cumulative effects of dust, vibration, abuse, environmental factors; i.i. it begins to <u>wear out</u>.

Sketch the idealized failure rate curve for software, Pressman Figure 1.3. Since software is not susceptible to those things that cause hardware to wear out, we might expect failure rate curve for SW to look like this.

Show the actual failure rate curve for software, Pressman Figure 1.4. Discuss:

i High failure rate early

ii As changes made to correct defects, new defects are introduced, causing "spike".

iii Before curve returns to steady state, more changes made & another spike.

iv Slowly, minimum failure rate rises (look at actual curve if spikes ignored); the SW is deteriorating <u>due to change</u>.

6. Configuration management - introduce term as transition to next lecture.

Maintenance involves dealing with change. Software maintenance must be dealt with in an effective controlled manner, consistent with the way we deal with software development. Otherwise errors will be introduced and costs will unnecessarily increase, and maintenance efforts and costs would continue to predominate. The management and control of software change is called <u>configuration management</u>, the topic of the next lecture.

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

Generate discussion by a series of leading questions. This topic presents a particularly good opportunity for discussion since analogies with hardware are familiar and plentiful. Student responses can inevitably lead

into the lecture material and provide an appropriate background for presenting the charts on maintenance efforts and costs.

<u>vocabulary introduced</u>:
maintenance
corrective maintenance
perfective maintenance (enhancement)
adaptive maintenance
failure rate
configuration management

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>:

L11OH1	Development Effort
L11OH2	Relative costs of phases of development
L11OH3	Relative maintenance cost by phase of development detected
L11OH4	Defects classified by time of introduction
L11OH5	Maintenance effort distribution
L11OH6	Input-Process-Output of maintenance

<u>handouts</u>:

<u>RELATED LEARNING ACTIVITIES</u>:
(labs and exercises)
Lab 009 - Feedback on design review presentations

Discussion questions (see procedure above).

Exercise: For the small project give an example of corrective, adaptive, and perfective maintenance. Make a list of enhancement requests that might be anticipated in the first year or so of operation. Categorize these as to type of maintenance.

<u>READING ASSIGNMENTS</u>:
Sommerville Chapter 28 (pp. 533-541)
Mynatt Chapter 8 (pp. 334-340)

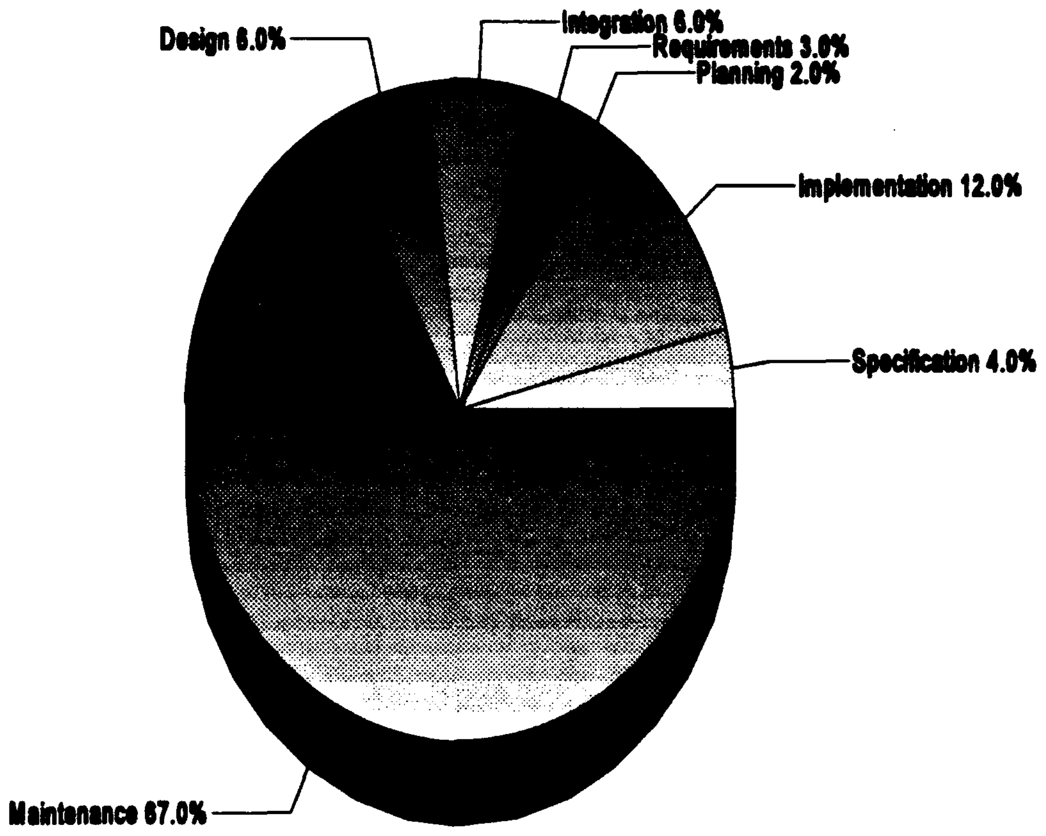
<u>RELATED READINGS</u>:
Booch Chapter 23 (pp. 422-423)
Booch(2) Chapter 19 (p. 403)
Ghezzi Chapter 2 (pp. 25-30)
Pressman Chapter 1 (pp. 7-22)
Schach Chapter 1 (pp. 8-12)

Development Effort *

Requirements	10%
Specification	10%
Design	15%
Code	20%
Module Test	25%
Integration Test	20%

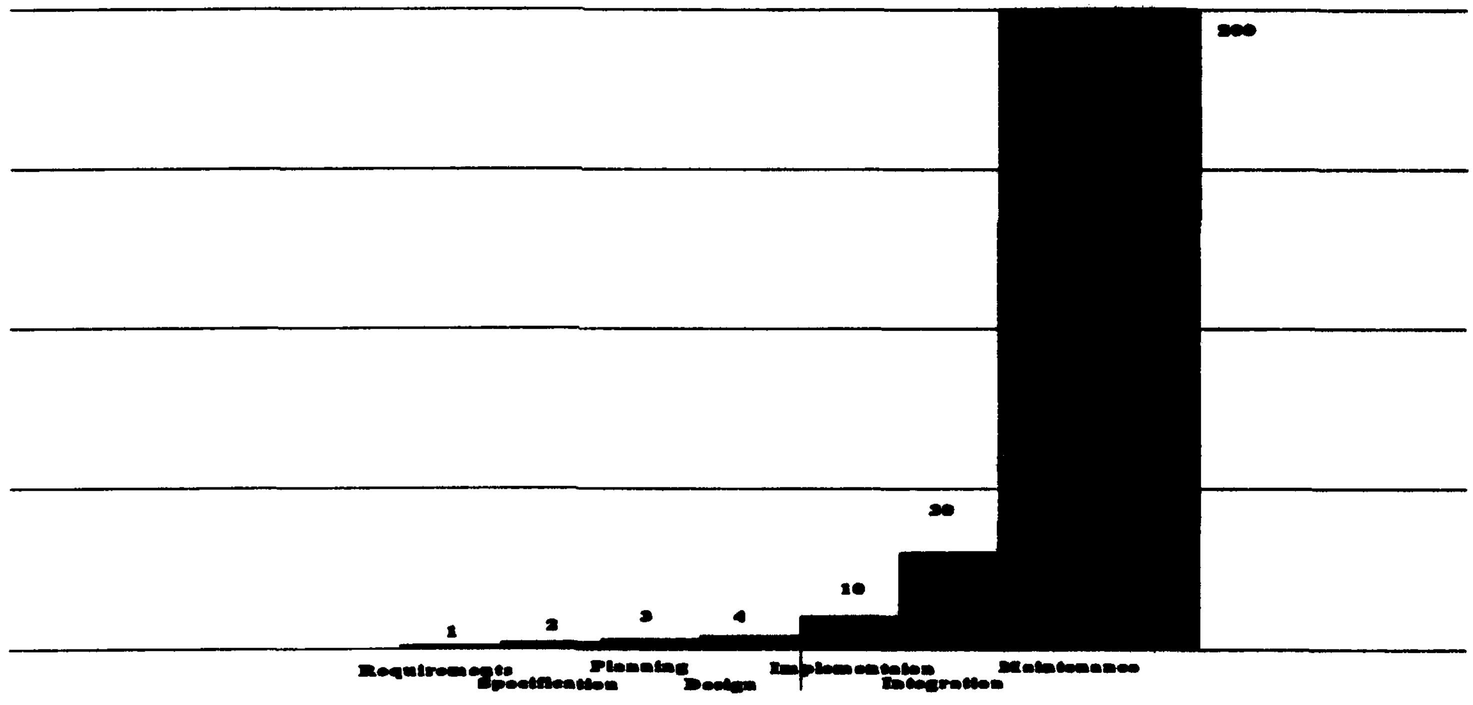
* does not include maintenance, which constitutes the largest portion of the software life cycle

Approximate Relative Costs
Phases of Software Production Process
Source: Schach, Software Engineering



Software Development Faults

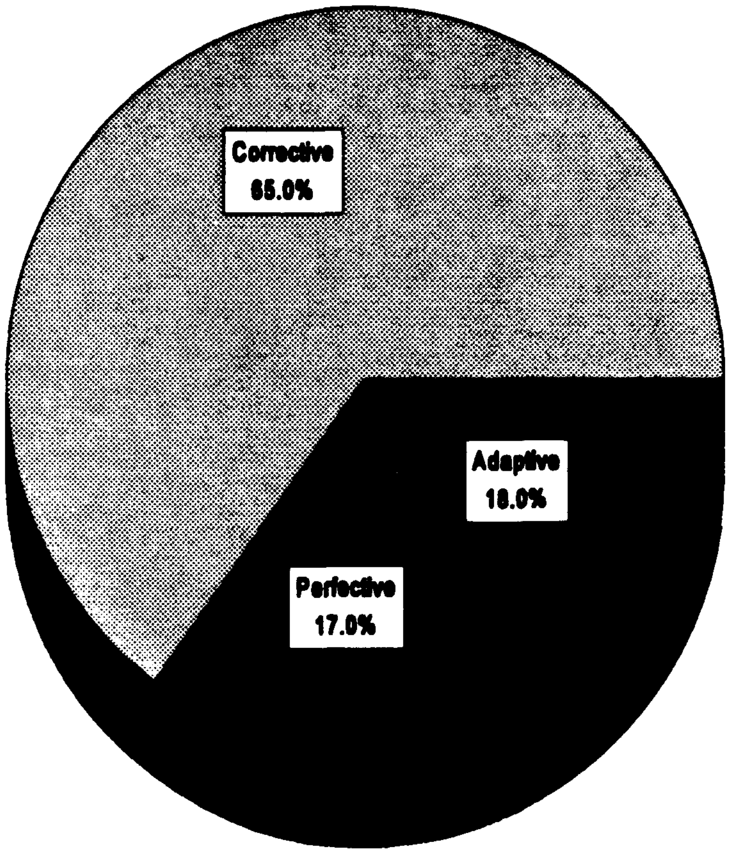
Relative Cost to Detect and Correct



Software Defects Classified By Time of Introduction

Inadequate or incorrect requirements definition	50%
Inadequate or incorrect design	15%
Errors in detailed design	10%
Coding Errors	20%
Other	5%

Maintenance Effort Distribution



Maintenance

Input

Software specifications
Preliminary design document
Detailed design document
Test plans and procedures
Source code
Proposed changes

Process

Incorporate changes
Retest system

Output

New program
New and/or revised documentation

LECTURE NUMBER:012

TOPIC(S) FOR LECTURE:

The importance of controlling disciplines in software development
Configuration management
Ways to implement configuration management

INSTRUCTIONAL OBJECTIVE(S):

1. **Recognize the role of configuration management over the entire life cycle.**
2. **Develop and evaluate a configuration management plan.**

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

L11OH4

We have just talked about maintenance. Maintenance is change to software that occurs after a system is developed. As we have seen, some errors are introduced into the software during the maintenance process. The development of software is a continuous process of change and affords developers a continuous opportunity to introduce errors into the system. Some would consider this opportunity for the introduction of error unacceptable. We cannot alter the nature of the development process, but if we manage and control the process of change we can restrict the opportunities for introducing error.

(Learning Label- Today we are going to learn ...)

In software engineering, the principles for controlling and managing change are called configuration management. Today we are going to look at the principles of configuration management and ways in which configuration management can be implemented.

CONTENTS:

1. Motivate the need for configuration management(CM) by discussing the simultaneous update problem and version control. CM is not just an issue about software. You have revised your small project requirements list several times. Ask the students what changes were made to the requirements list and whether they all were certain that all would have been approved by the customer? Did you check with the customer?

    a. Multiple people working on a large project can have different understandings of what the system is supposed to do or may make small changes which do not work well with other parts of the system. Using the test plan L9OH6 remind the student that the test planner made a change which required the KoFF system to track expired cards. Was this information communicated to the designers, coders or even to the customer? What is the likelihood of this change made by the test planner ever getting implemented? What can be done to assure that these kinds of changes are acceptable and will get implemented? There is a need for control management and communication of change.

b. There are multiple sources and reasons for change requests. These occur throughout the development process as well as after system development. Talk about change requests as desired improvements in the system. As the customer better understands the system he/she sees new and improved ways that the system could be developed. Changes also come from the developers' improved understanding of the requirements, and changes in the environment while a system is being developed. This can lead to chaos unless carefully managed.

c. Sometimes systems are developed in different versions, e.g., DOS 2.2 - DOS 6.0. Each version of each system has to be tracked and maintained. Versions are not always sequentially developed, as was DOS 4, DOS 5, and DOS 6. Sometimes multiple versions of the same system are developed concurrently to fit on different hardware platforms, e.g., UNIX for DEC and IBM can be developed at the same time. The requirements for these systems are different, and one must track and maintain multiple versions of the "same" product.

d. Talk about baselining as a technique for limiting or controlling this chaos. How is baselining done? Be sure to emphasize that this involves formal review and agreement by all concerned parties. Once an item is baselined it is under change control and can only be changed by formal change control procedures. What is it that gets baselined. Proposed changes to baselines are called Change Requests(CR).

2. Methods of CM require a plan, a well defined process, and a manager to carry out the plan.

a. Ask the class what things they need to keep constant to develop a system. List these on the board. Discuss them as Configuration Items (CI). Display overhead of standard configuration items L12OH1. Work through each item talking about those items which are new to them. Be sure to emphasize that any change requires approval and communication of the change as well as updating the affected documents. Another function of CM is to maintain consistency between the documentation of a system and the system itself.

b. CM is complex and requires a plan to be sure it is executed. Display overhead L12OH2. Go over the contents of the IEEE CM Plan. Briefly go over the management issues, such as how configuration management relates to other organizations. Discuss overhead item 2.d which includes naming conventions for components and how CRs will get processed. Distribute handout L12HD1 as an example of a portion of a student-produced CM plan.

c. *To maintain control, baselined configurations items are* sometimes placed in a special electronic library. Permission to change or modify CIs is gained through a CR approval process. CRs are generally approved by groups called <u>Configuration Control Boards (CCB)</u>. Discuss some standards used to decide the approval of CRs; e.g, functional need, cost versus benefit analysis, impact on other modules, politics (the president of the company "just wants it").

d. There are several virtues of CM which include reducing the number of errors generated, minimizing the use of storage, giving visibility to system development progress each time a new CI is baselined and reducing the time and effort costs associated with uncontrolled change.

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

At this point in the course students have likely experienced uncontrolled changes within their small projects. Some of these have also likely caused problems. Their own "war stories" can serve to enhance their interest and appreciation of the necessity for configuration management. The primary teaching technique consists of using lecture and overheads with frequent reference to problems they have encountered in their small project teams.

<u>vocabulary introduced</u>:

configuration management (CM)
configuration item (CI)
baseline
discrepancies versus changes
configuration control board (CCB)
change request (CR)

INSTRUCTIONAL MATERIALS:

overheads:

L12OH1	Configuration items
L12OH2	IEEE Model for a configuration management plan

handouts:

L12HD1	Student configuration management plan

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 010 - Feedback on CI-5, test plans, and test cases.Small project team preparation for team acceptance test presentations

READING ASSIGNMENTS:

Sommerville Chapter 29 (pp. 551-564)
Mynatt Chapter 8 (pp. 336-340)

RELATED READINGS:

Ghezzi Chapter 7 (pp. 403-408)
Pressman Chapter 21 (pp. 693-708)
Schach Chapter 4 (pp. 87-93)
James Tomayko, "Support Materials for Software Configuration Management," Support materials, SEI_SM_4_1.0
IEEE Standard for Software Configuration Management Plans, IEEE Std 828-1983

Configuration Items

Requirements Documents
- Client Request
- Requirements List
- Analysis Documents
- Revision History
- Revision requests and approvals

Design Documents
- Preliminary Design Documents
- Preliminary Design Review Documents
- Detailed Design Documents
- Detailed Design Review Documents
- Revision History
- Revision requests and approvals

Code Documents
- Source code modules
- Object code modules
- Compiler used
- System build plan

Other Documents
- Test Plan
- Test Cases
- Test Results
- User Manual
- Referenced Documents

IEEE Model for a CONFIGURATION MANAGEMENT PLAN

1. Introduction
   a. Purpose
   b. Scope
   c. Definitions and acronyms
   d. References

2. Management
   a. Organization
   b. Configuration management responsibilities
   c. Interface control
   d. Implementation of plan
   e. Applicable policies, directives and procedures

3. Configuration management activities
   a. Configuration identification
   b. configuration control
   c. Configuration status accounting
   d. Audits and reviews

4. Tools, techniques, and methodologies

5. Supplies Control

6. Records collection and retention

Configuration Management Plan

```
+-----------------------------------------------------------
|   PROJECT:               Third Eye Project
|   FILE NAME:        CM_PLAN.DOC
|   DOCUMENT NAME:        Configuration Management Plan
+-----------------------------------------------------------
|   PURPOSE:
|        This document describes the responsibilities of
|        Configuration Management.
+-----------------------------------------------------------
|   MODIFICATION HISTORY:
|        WHO:                         REV:              DATE:
|        Kellie Price
|          * Created initial revision of document.
+-----------------------------------------------------------
```

Computer and Information Sciences

Third Eye Project

Configuration Management Plan

Kellie Price

Table of Contents

1. PURPOSE

The Configuration Management Plan defines the Configuration Management (CM) policies which are to be used in the Third Eye Project. It also defines the responsibilities of the project configuration manager.

2. MANAGEMENT

2.1 CONFIGURATION MANAGER RESPONSIBILITIES

The first responsibility of the configuration manager is to develop and implement this Configuration Management Plan.

Throughout the project, the configuration manager will report directly to the customer. It is the configuration manager's responsibility to ensure that the project is implemented in a straight-forward and well-defined manner according to the customer's specifications and standards established by Configuration Management for this project.

2.2 ORGANIZATION

This project will be divided into 7 teams as follows:
(Refer to CM_TEAMS.DOC for the specific team assignments)

NOTE: All of the documents required of each team below are listed in the file CM_DOCS.DOC.

2.2.1 REQUIREMENTS TEAM

The Requirements Team is responsible for communicating with the customer in order to determine and well-define the software system requirements. The documents required of the Requirements Team are:

* Narrative description of system
* List of requirements (acceptance criteria)
* Context Diagram
* A series of leveled Data Flow Diagrams
* Data Dictionary
* Process Specifications

2.2.2 USER MANUAL TEAM

The User Manual team is responsible for producing all user documentation for the system. The documents required of the User Manual Team are:

* Preliminary format of user manual
* User Manual

2.2.3 TEST PLAN TEAM

The Test Plan team is responsible for designing subsystem and system tests. The documents required of the Test Plan Team are:

* Test plan

2.2.4 PRELIMINARY DESIGN TEAM

The Preliminary Design team is responsible for creating a preliminary design structure of the system based on the software system requirements. The documents required of the Preliminary Design Team are:

* An Object Model:
    * Complete object diagram
    * Class dictionary
    * Object-Requirements traceability matrix
* Ada Specifications for each object class

2.2.5 DETAILED DESIGN TEAM

This team is responsible for creating algorithms to implement the system structure. The documents required of the Detailed Design Team are:

* Data Structure Design using a data structure dictionary

* Algorithm Design using Nassi-Shneiderman models

* An object attributes and object operations traceability matrix

2.2.6 CODE & UNIT TEST TEAM

The Code & Unit Test team is responsible for producing source code for the algorithms produced by the Detailed Design Team, integration of the modules to produce a

working system. The documents required of the Code & Unit Test Team are:

* Source code

2.2.7 TESTING TEAM

The Testing team is responsible for implementing the tests in the test plan and using them to test the system. The documents required of the Testing Team are:

* Test data
* Documented test results

3. CONFIGURATION MANAGEMENT ACTIVITIES

3.1 C.M. REQUIREMENTS DOCUMENTS

The configuration manager has provided documentation to assist the teams in meeting the C.M. requirements. This documentation is in a series of files which are available on the project file server. The C.M. requirements defined in these files are as follows:

DESCRIPTION	FILENAME
* Documents required by C.M.	CM_DOCS.DOC
* Document header info	CM_HEADR.DOC
* Document naming conventions	CM_NAMES.DOC
* Document format & standards	CM_FORMT.DOC
* Change request form format	CM_CHREQ.DOC
* Configuration item request procedure	CM_CIREQ.DOC
* Configuration item access procedure	CM_ACESS.DOC
* Configuration item change process	CM_CHPRO.DOC
* Configuration item baseline process	CM_BASLN.DOC

3.2 C.M. CONTROL

The configuration manager will provide the teams and team members controlled access to their respective configuration items. In order to have access, however, the teams and/or team members must provide the configuration manager with a written request for any desired configuration items as defined in the file CM_CIREQ.DOC.

4. CONFIGURATION MANAGEMENT RECORDS

All BASELINED Configuration Items and documents will be maintained on the project file server in a directory structure as defined in the file CM_FILES.DOC.

4.1 C.M. FILES

All Configuration Management files (including the requirements files listed in section 3.1) are listed below:

DESCRIPTION	FILENAME
* Configuration item access procedure	CM_ACESS.DOC
* Configuration item baseline process	CM_BASLN.DOC
* Configuration item change process	CM_CHPRO.DOC
* Change request form format	CM_CHREQ.DOC
* Configuration item request procedure	CM_CIREQ.DOC
* Original customer request	CM_CRQST.DOC
* Documents required by C.M.	CM_DOCS.DOC
* C.M. file directory structure	CM_FILES.DOC
* Change request form	CM_FORM.DOC
* Document format & standards	CM_FORMT.DOC
* Document header info	CM_HEADR.DOC
* Document naming conventions	CM_NAMES.DOC
* Document page header	CM_PGHDR.DOC
* Configuration Management Plan	CM_PLAN.DOC
* Software Project Management Plan	CM_SPMP.DOC
* Project team organization	CM_TEAMS.DOC

<u>LECTURE NUMBER</u>: 013

<u>TOPIC(S) FOR LECTURE</u>:
Ada and maintenance

<u>INSTRUCTIONAL OBJECTIVE(S)</u>:

1. Understand technical aspects of a language which support maintainability.
2. Recognize the language features of Ada which support maintainability of a software system.

<u>SET UP, WARM-UP</u>:
(How involve learner: recall, review, relate)
(Learning Label- Today we are going to learn ...)

<u>CONTENTS</u>:

L13OH1

1. Consider and discuss three technical factors that affect the maintainability of a software system: completeness and consistency, understandability of program and its documentation, and modifiability of the system.

   a. Software systems in industry which have high maintenance costs are systems which are long lived, are complex, and must adapt to changing requirements and changing hardware. The technical factors in L13OH1 are needed to make such software systems more maintainable.

   b. Completeness and consistency of system documentation are independent of the implementation language. I a system is complete and consistent then any changes made to the program are reflected in the requirements definition, design documents, test plans, etc. The implementation of such completeness and consistency is dependent on project management.

   c. Understandability of a program and its documentation is important so that changes can be readily made. The understandability of the program is a language dependent technical factor.

   d. Another language dependent technical factor is the modifiability of the system. A system is modifiable if a change made to one part of the system affects that part and that part only. The language, therefore, needs to provide the language features which allow a system to be built of stand-alone components which do not interfere with other system components.

L13OH2

2. Discuss specific language features of Ada that support maintainability.

   a. Named association, illustrated in overhead L13OH3, provides the ability to assign names to formal parameters and to use these names in actual parameter association.

   b. Overloading provides the ability to define multiple meanings to individual operators and procedure/function names. Overloading allows expressions of user-defined types to be written using familiar notation. An example is given on overhead L13OH4.

   c. Discuss packages and how they promote modifiability in a system because subsystems can be built which act as stand-alone components.

   d. Discuss the separation of interface specifications and bodies which allows the details of implementation to be abstracted away. This feature promotes understandability because only the essential interface information is shown and the implementation details are hidden away in the unit's body. Discuss separate compilation of the specifications.

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

<u>vocabulary introduced</u>:

named association of parameters
overloading
separate compilation
Ada package specification and body

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>:

L13OH1	Maintenance
L13OH2	Ada and Maintainability - features
L13OH3	Ada and Maintainability - example of named association of parameters
L13OH4	Ada and Maintainability - example of overloading

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)
Lab 011 - Presentation of customer request for extended project

READING ASSIGNMENTS:
None

Maintenance

Technical factors that affect the maintainability of a software system

1. Completeness and consistency of system documentation
2. Understandability of program and its documentation
3. Modifiability of the system

Ada features which support maintainability:

Named association of parameters

Overloading

Packages

Separation of interface specifications

ADA AND MAINTAINABILITY

An example of named association of parameters

procedure definition:

```
procedure compute_speed
 (old_x_coord, old_y_coord, old_z_coord,
 new_x_coord, new_y_coord, new_z_coord:in real;
                              speed : out real);
```

procedure invocation:

```
compute_speed
 (13.459, -18.634, 28.775,
 24.762,  -98.628, 45.350,
                  value);

compute_speed (old_x_coord => 13.459,
               old_y_coord => -18.634,
               old_z_coord => 28.775,
               new_x_coord => 24.762,
               new_y_coord => -98.628,
               new_z_coord => 45.350,
               speed => value);
```

AN EXAMPLE OF OVERLOADING

```
type complex_number is
     record
          ...
     end record;

-------------------------------
procedure multiply_complex
     (c_1, c_2 : in complex_number;
      c_3 : out complex_number);


multiply_complex (c_stream,
                  c_pause,
                  c_dir);

-------------------------------
function "*" (c_1,
              c_2 : in complex_number)
             return complex_number;


c_dir := c_stream * c_pause;
```

LECTURE NUMBER: 014

TOPIC(S) FOR LECTURE:
Software life cycle models

INSTRUCTIONAL OBJECTIVE(S):

1. Understand the concept of a software life cycle.
2. Understand that a variety of life cycle models exist.
3. Distinguish between several life cycle models, including waterfall, prototyping, and spiral models, and know the strengths and weaknesses of each.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

Refer back to the introductory lectures on the various activities in software development. At that time we briefly described the classic life cycle model and discussed some of its strengths and weaknesses. We're going to review those today, with a little more detail, and then discuss some other life cycle models and look at their strengths and weaknesses.

(Learning Label- Today we are going to learn ...)

We will be looking at process here, as distinguished from product. An organization that can devote attention to process as well as product has achieved some measure of maturity.

CONTENTS:

1. Concept of a life cycle model - a series of phases through which the software product progresses is a software process model, or a software life-cycle model.

   a. Fundamental (generic) steps in software development.
      Review the fundamental activities: requirements analysis, specification, design, implementation, testing, maintenance. Emphasize that in general there are not the sharp boundaries between the different life cycle activities that are implied when we differentiate between them in order to discuss them.

      An artificial boundary between these activities is indicated by the development of a milestone document.

   b. *Ad hoc* methods (e.g. the "build-and-fix model"), though used in places and even referred to as models, are not really life cycle models but instead demonstrate the absence of a model.

We are considering here systematic approaches that represent repeatable processes.

c. Discussion question: Why is it important for a software development organization to have a well defined process?
   i To define activities that are to be carried out and deliverables and milestones associated with each
   ii To introduce consistency among projects
   iii To provide checkpoints for management control and for go/no go decisions
   iv Increasing software backlog due to increased demand for software services - Use this to further emphasize the need for an effective process for software development.

2. Classic life cycle model (Waterfall)

a. Description - L14OH1 - Fig 1.7 Pressman
This is a systematic, sequential approach similar to traditional engineering cycles and note that there is feedback between life cycle phases. There are verification activities included in each phase.
Consider the phases:
   i System engineering - Typically the software is part of some larger system and this phase involves establishing requirements for the entire system and then allocating some subset of this to software.
   Use the small projects as an example. Identifying the system context (scope) and interfaces with external entities. This "external" view was necessary before devoting attention to the software system.
   ii Software requirements analysis - This involves the extraction and clarification of requirements for the software system.
   Requirements and specification of requirements are documented and reviewed with customer and are baselined as a software configuration item. A preliminary test plan, based on the requirements list, is also developed and baselined here.
   iii Design - The design is documented and reviewed with customer and becomes part of the software configuration.
   iv Coding and component testing -
   The results of component coding and component testing are documented and checked against the original test plan and design. Code, test data & test results become a part of the software configuration.

v Integration and system testing
After successful test results are achieved, the test plans and results are reviewed with customer and become part of the software configuration. This testing is called acceptance testing.

vi Maintenance - reapplies each of the preceding life-cycle phases to an existing system.

vii The waterfall model is the oldest and most widely used software life cycle model. Schach presents a variation on the waterfall model which includes verification at every phase. If verification is included as an integral part of each phase then it is clear that this is not an ad hoc model. L14OH2 Here a phase is not complete until the documentation is reviewed and approved and placed under configuration control.

b. Strengths of waterfall model
   i Disciplined - requires documentation and verification at each phase.
   ii Documentation-driven.
   iii Widely used.
   iv Far better than *ad hoc* approach.

c. Weaknesses of waterfall model

   i Even with feedback, the model is essentially sequential and, for many projects, that is not realistic. It is often difficult for the customer to state all requirements explicitly at the start of a project.

   ii Documentation-driven - While listed as an advantage, it can also be a disadvantage. The documentation is often not adequate for the customer to understand and even though he/she is signing off at each phase, he/she may not really understand the system from that documentation.

   iii A working version of the system is not available until the later phases. This problem is addressed in the prototype model.

3. Prototyping model - Introduce this by discussing how the weaknesses of the waterfall model might be addressed.
Prototyping is the creation of a functionally- equivalent model of the system, or a subset of the system, that can be demonstrated to the customer. This demonstration can take several forms; a paper model (e.g., showing user screens and reports), an existing system that

provides similar functionality and/or interface, or a skeleton version of the final system. Note different forms the model can take.
L14OH3

a. Prototyping begins with requirements gathering (involving developers and customers) followed by a rapidly developed design and construction of a prototype. The customer evaluation of this working version, leads to a clarification of the requirements.

b. Discuss Brooks' observations, made in 1975, in The Mythical Man-Month. Read from chapter 11 (Plan to Throw One Away), p 116. L14OH4

c. Discuss problems associated with prototypes.
   i Customer sees working version early and may confuse the prototype and the final system, thus expecting a finished product in an unreasonable amount of time. This may result in pressure to turn the prototype into the real system and, in turn, would result in an inadequate (unmaintainable, untested, etc) product.
   ii Implementation compromises are made to get a working prototype early. Developer, for a variety of reasons, may forget these compromises.

4. Spiral model - The spiral model was developed by Barry Boehm to incorporate the advantages of prototyping into the waterfall model.

   a. Background - The development of application software for real customers always involves elements of risk. What are some risks?
      i It may not be clear that some requirements are attainable (response times, dependency on new technology, new theory, schedule requirements, necessary personnel/expertise not available, requirements not testable, etc)
      ii Dependencies on other systems or hardware which are beyond the developed control (relate to small projects)

   Note that one way to reduce risk would be to build a prototype in order to resolve risks early in the project. Review the weaknesses of the previously discussed models (waterfall and prototyping) as an introduction to the spiral model. This gives an historical perspective and emphasizes the evolution of process models for large software systems.

The intent of the spiral model is to encompass the best features of the waterfall and prototyping approaches and, at the same time, incorporate risk analysis. In the spiral model, risks are identified and an attempt is made to resolve them through the use of prototypes and other means [illegible] the students how this relates to the projects they are wo [illegible].

L14OH5

b. Major activities represented by the 4 quadrants.

i Top-left quadrant: Planning - determination of objectives, alternatives, constraints; requirements

ii Top-right quadrant: Risk analysis - analysis of alternatives and identification/resolution of risks

iii Bottom-right quadrant: Develop this portion of the system following the most appropriate process model.

iv Bottom-left quadrant: Customer evaluation - assessment of how the product of this phase relates to the initial plan and the product of the previous phase. This is followed by planning the next iteration of the spiral.

c. With each iteration around the spiral, progressively more complete versions of the software are built. During the first trip of the spiral, objectives are established, alternatives for achieving those objectives and constraints are identified. Based on the risk evaluation a development model is chosen. (For example, if risk analysis indicates sufficient uncertainty in requirements, prototyping may be used). Finally the results are evaluated and the next trip around the spiral planned. Key elements of this model are the assessment of risk at regular intervals and the initiation of actions to address/minimize the risks. Thus high risks would be addressed early. Risk analysis is done before each cycle and an assessment is done at end of each cycle.

d. Strengths of spiral model

i Emphasis on risk identification, assessment, and resolution.

ii Considered by many to be the most realistic approach to development of large scale software systems.

iii Incorporates advantages of waterfall and prototyping models.

e. Weaknesses of spiral model

i Applicable for large-scale systems only.

ii Requires risk assessment expertise.

5. There are many life cycle models possible; we've looked at some representative models and their strengths and weaknesses. Ask the students what factors might determine the particular life cycle model an organization chooses for a project.

   a. Stability of requirements

   b. Problem domain

   c. Risk: economic, schedule, feasibility, safety, ethicality

   d. Organization and expertise available

PROCEDURE:

teaching method and media:

vocabulary introduced:

software process model/life cycle model
process (vs product)
repeatable process
waterfall model
prototyping model
spiral model
risk, risk analysis

INSTRUCTIONAL MATERIALS:

overheads:

L14OH1	Waterfall model (Fig. 1.7, Pressman)
L14OH2	Waterfall model (Schach, p.50)
L14OH3	Prototyping (Pressman, p. 28)
L14OH4	Quote from Brooks, The Mythical Man-Month
L14OH5	The spiral model (Sommerville, p. 15)

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab012	Project Team Organization

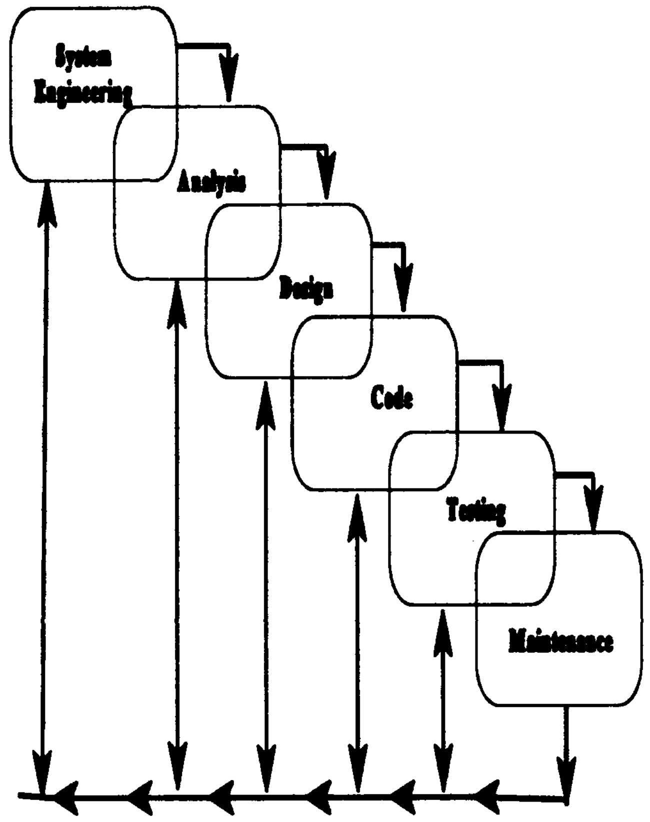
READING ASSIGNMENTS:

Sommerville Chapter 1 (pp. 5-18)
Mynatt Chapter 1 (pp. 12-27)

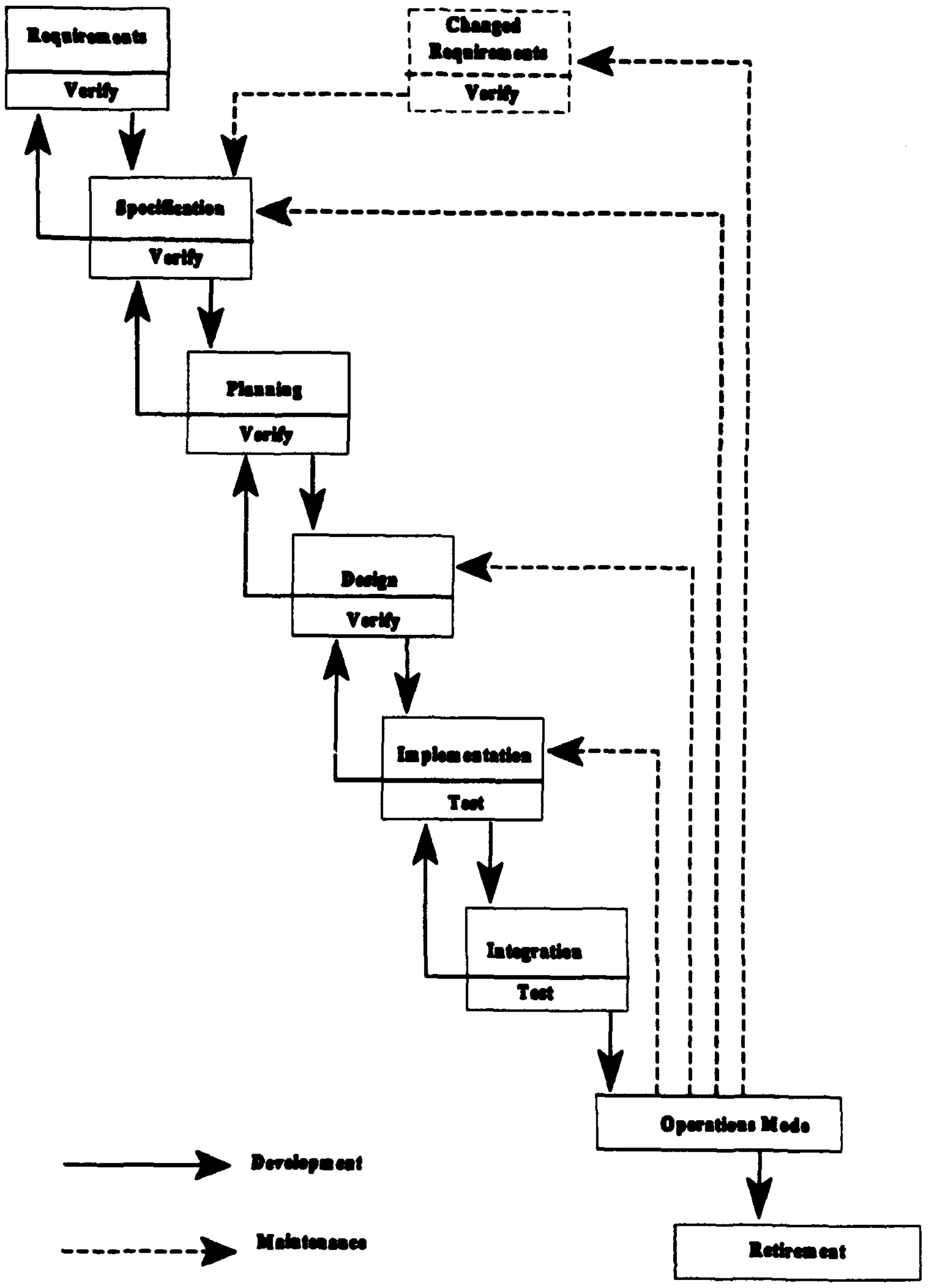
__RELATED READINGS__:

Ghezzi Chapter 7 (pp. 357-383)
Pressman Chapter 1 (pp. 24-36)
Schach Chapter 3 (pp. 47-66)

Classic Life Cycle (Pressman) Waterfall Model



Waterfall Model (Schach)



PsuedoCode Example

Policy for ordering new stock

```
For each New_Stock_Request, do the following:

    1.  Search for an Authorization_Form with
        Reference_Number equal to the
        Request_Number on the
        New_Stock_Request.

    2.  If there is no match
        Then Discard this New_Stock_Request
        Else

            a.  Write a Purchase_Order for
                Ordered_Item
            b.  From Supplier_Catalog, select a
                Supplier who carries the
                Ordered_Item
            c.  Copy Supplier_Name and
                Address to Purchase_Order
            d.  Copy Purchase_Order_Number to
                New_Stock_Request
            e.  File New_Stock_Request with
                Authorization_Form
```

Process
Data Dictionary Form

Process Name: *Pay Union Dues*
Purpose: Every four weeks produce the union deduction register and consolidate employee union deductions to output the union check, and the Accounting union summary, and expense entry.

Input Data Flows: *Employee Union Deduction*

Output Data Flows:*Union Expense Entry*, *Union Summary*,*Union Check*, *Union Deduction Register*

Process Description:

```
Sort Employee Union Deduction by Union Number,
 Employee Name, Week Ending Date

For each Employee Union Deduction
    Output Union Deduction Register
    IF Union Number changes
        Output Union Check
        Output Union Summary
```

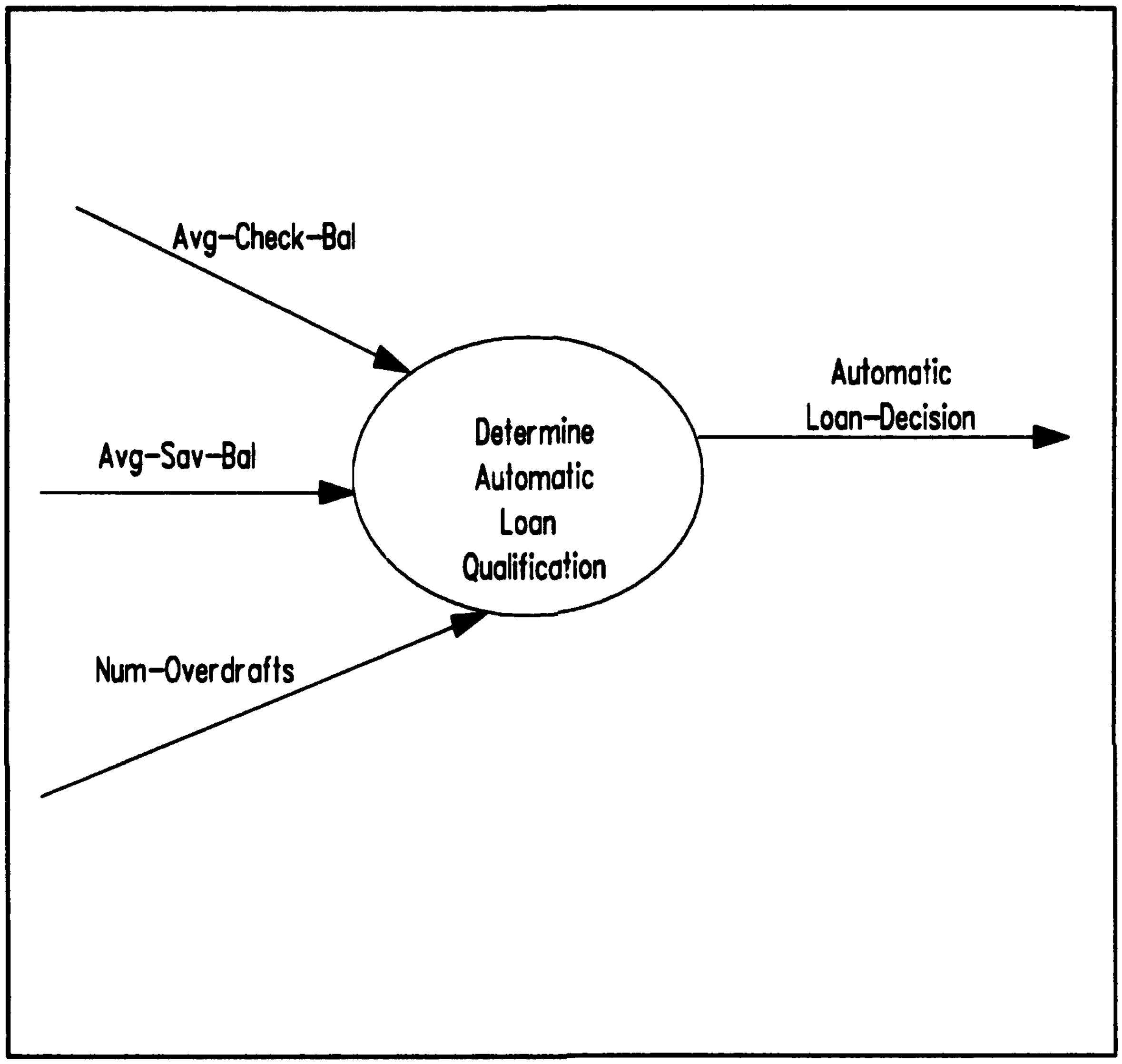
Shelly-Cashman, Figure 4-35

Structured Requirements Specification

Function:	Check_card_validity
Description:	This operation must ensure that the card input by a user has been issued by a subscribing bank, is in date, contains appropriate account information, and includes details of the date and amount of the previous cash withdrawal.
Inputs:	Bank-identifier, Account-number, Expiry-date, Last-transaction-date, Last-transaction
Source:	Input data is read from the card magnetic stripe
Outputs:	Card-status = (OK, invalid)
Destination:	Auto-teller. The card status is passed to another part of the software.
Requires:	Bank-list, Account-format, Today's-date
Pre-condition:	Card has been input and stripe data read
Post-condition:	Bank-identifier is in Bank-list and Account-number matches Account-format and expiry-date >= Today's-date and Last-Transaction-date <= Today's-date and Card-status = OK or (if any of these tests fail) Card-status = invalid

Sommerville, Figure 5.1

Translation of Bank Loan Qualification Policy into a Data Flow Diagram

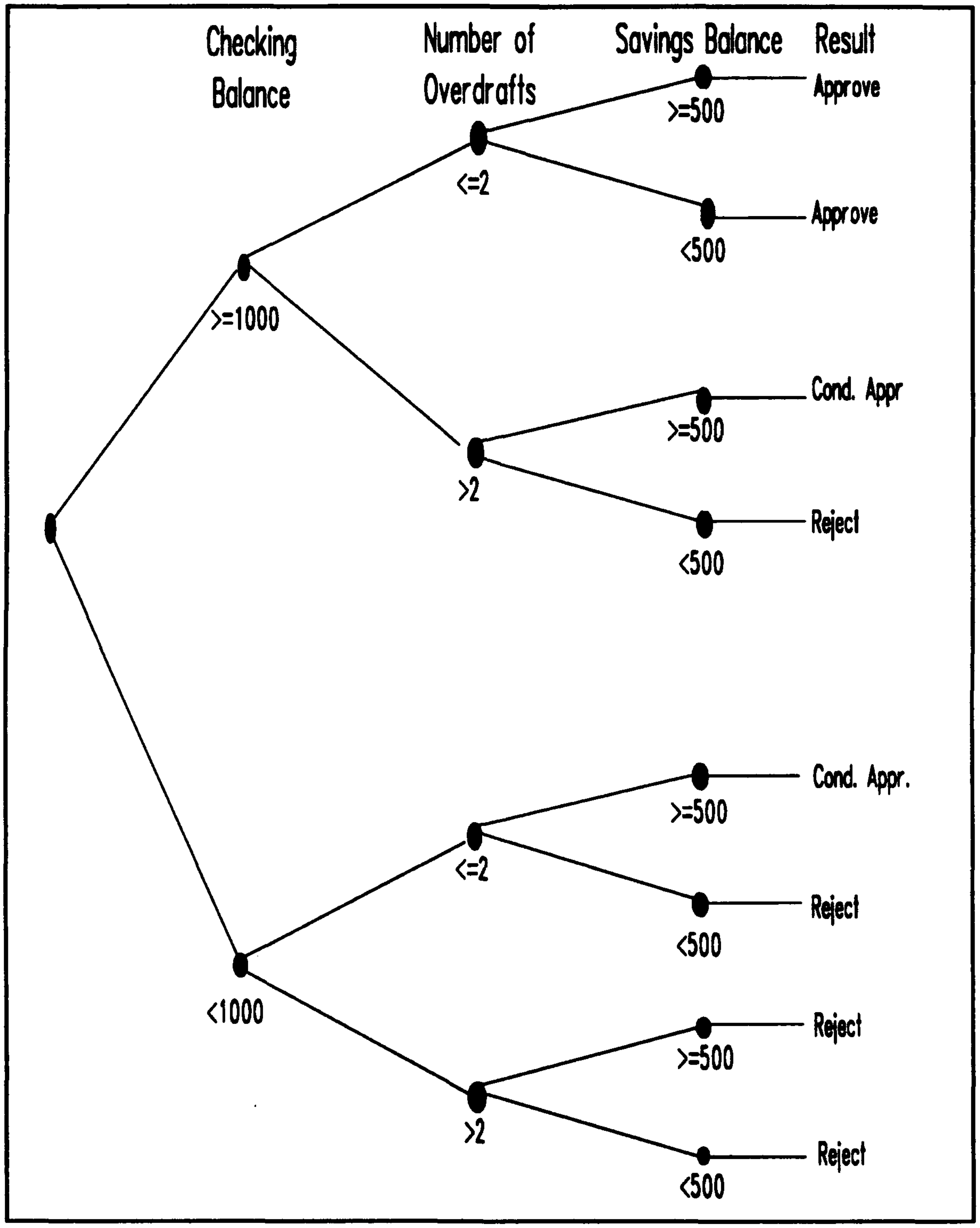


The customer is approved if he or she has maintained an average monthly checking balance of at least $1,000 for each of the last three months and has averaged no more than two overdrafts per month. Customers meeting only one of these conditions but maintaining an average savings account balance of at least $500 for each of the last three months receive conditional approval with an automatic loan limit of $500.

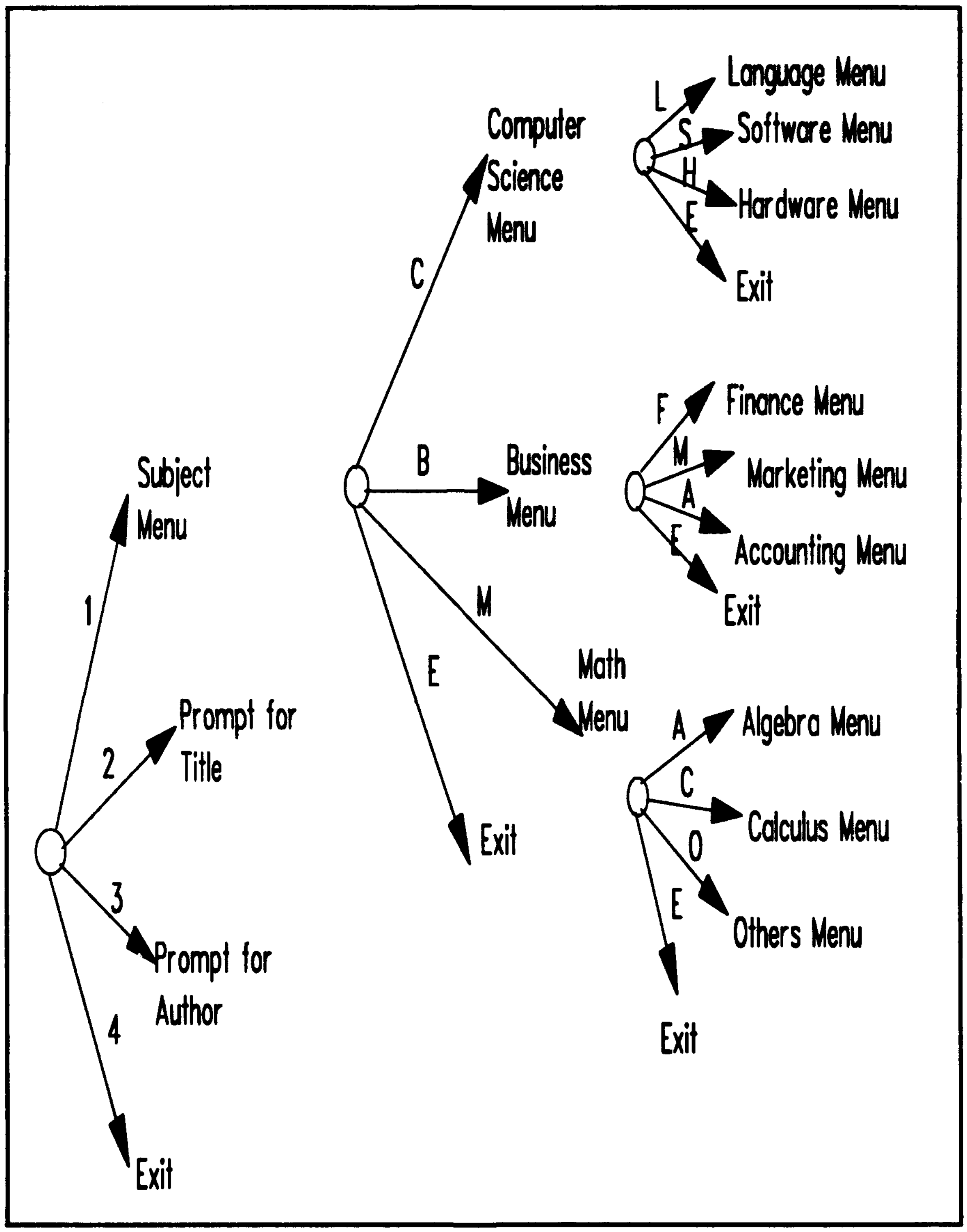
Decision Table Covering Policy For Automatic Loan Qualification

Avg Ck Bal >= 1000	Y	Y	Y	Y	N	N	N	N
Num overdrafts <= 2	Y	Y	N	N	Y	Y	N	N
Avg Sav Bal >= 500	Y	N	Y	N	Y	N	Y	N
Approve	X	X						
Cond. Approve			X		X			
Reject				X		X	X	X

Decision Tree Expressing A Bank's Policy Concerning Loan Qualifications

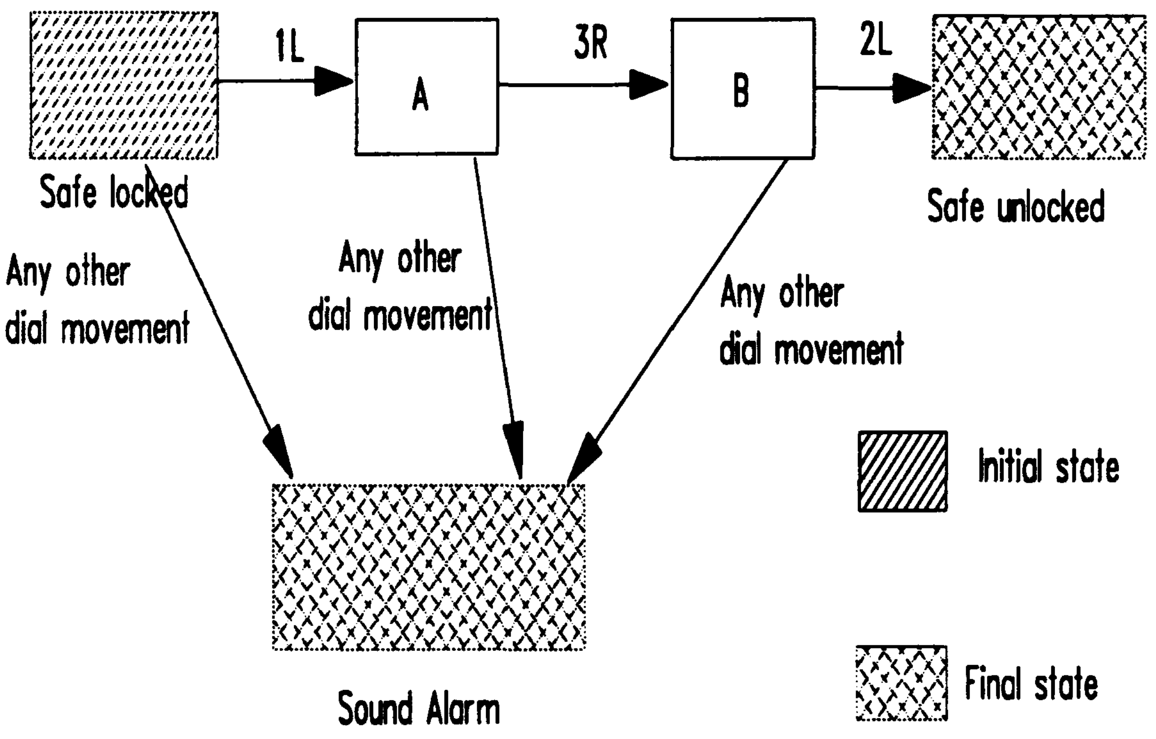


Decision Tree Showing the Flow of Control



Eliason,p. 390

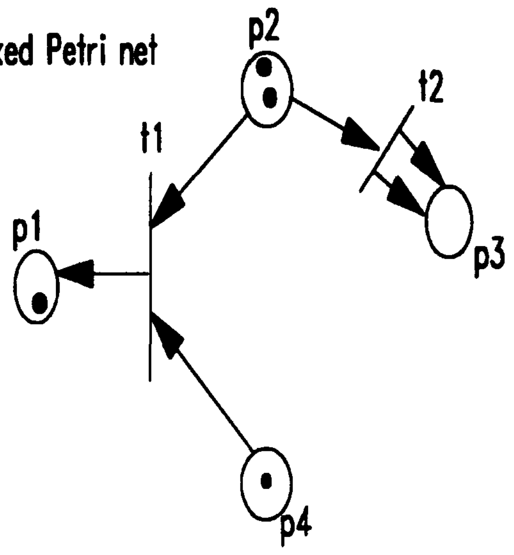
Finite State Machine Representation Of Combination Safe



Schach, Figure 7.8

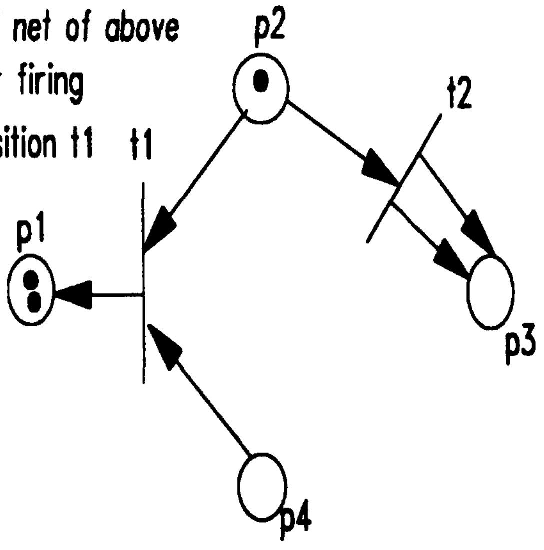
Petri Nets

Marked Petri net



t = transition
p = place

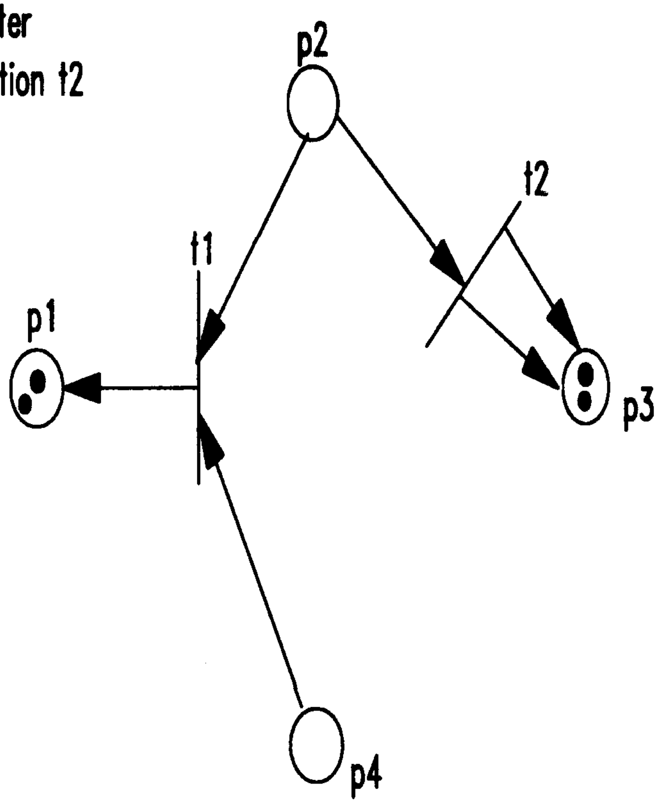
Petri net of above
after firing
transition t1



Schach, Figures 7.14, 7.15

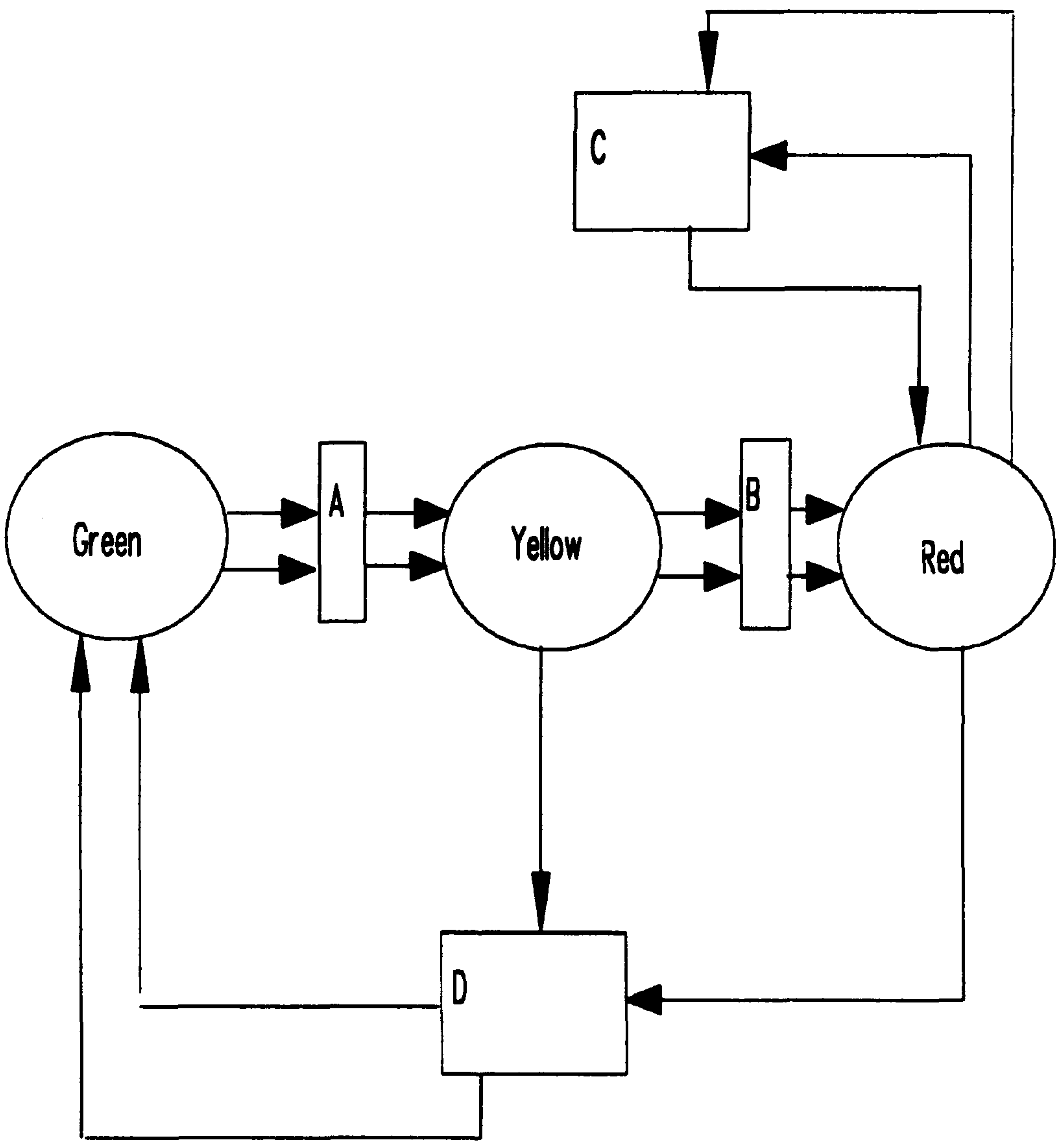
Petri Nets

Petri net after firing transition t2

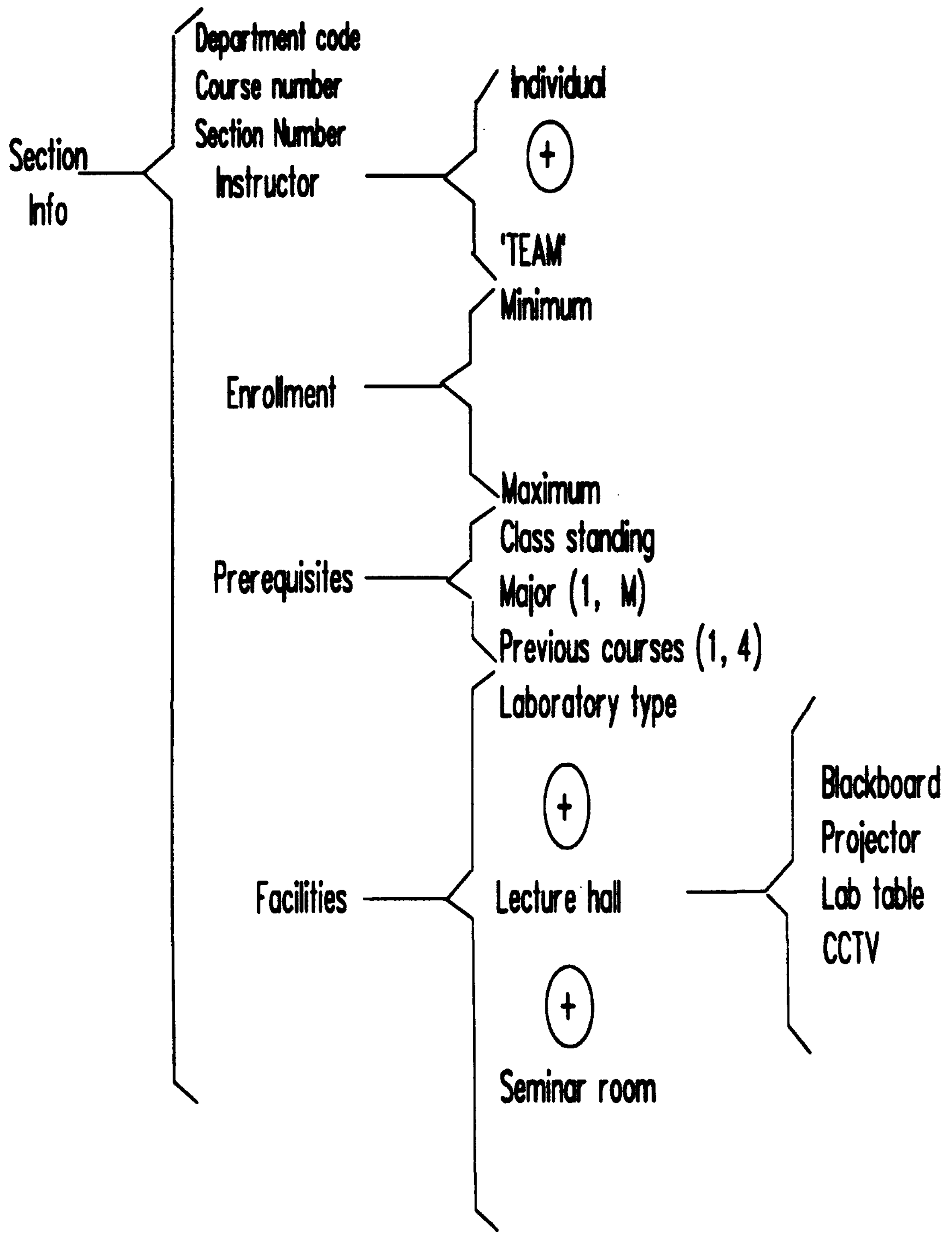


Schach, Figure 7.16

Execution of a Petri Net



A Warnier Diagram



Jones, p. 198, Figure 4-25

LECTURE NUMBER: 024

TOPIC(S) FOR LECTURE:
Transform analysis
Transaction analysis

INSTRUCTIONAL OBJECTIVE(S):

1. Understand transform analysis and its application.
2. Understand transaction analysis and its application.
3. Recognize when transform analysis is appropriate and when transaction analysis is appropriate.
4. Understand the process of refining first-draft structure charts produced by transform analysis or transaction analysis considering design criteria such as coupling and cohesion.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

In your first project you had to develop structure charts, without using any well-defined method.

(Learning Label- Today we are going to learn ...)

Today we're going to look at two methods of developing structure charts from data flow diagrams.

CONTENTS:

1. Transform analysis is a set of design steps that can be applied to map a data flow diagram into a structure chart. Theoretically, transform analysis can be applied to any system but it is more appropriately applied to specific types of systems. The input to transform analysis is the DFD model; the output is a first-draft structure chart.

2. The first step in transform analysis is to identify the central transform in a DFD.

   a. L24OH1
   For many systems input comes into the system from the external world (keyed in by user, signals from a sensor), is transformed by some internal process(es), and results are output (printed report, graphics display, signal to external device). The overhead represents this as a function of time as the system performs its task. Note that information flows from its external entry into the system (the afferent streams), into some internal representation where the essential transformation takes place, and then flows out (the efferent streams)to the external world. Tracing the input, at some point the input data

ceases to be input and becomes internal data, i.e. it ceases being raw data (through verification, editing, filtering, ... operations). Similarly, at some point the internal data is transformed into output data. The key to transform analysis is identifying the part of the system where the "essential transform" takes place; the boundaries at which raw input data has been transformed into essential data, and at which essential data becomes output data but still has to be formatted, refined, etc before it can be output from the system.

L24OH2
For example, consider this master file update (briefly review or explain a traditional master file update). Where does the essential transformation occur? Points A and B in the overhead are the beginning of the afferent streams; at these points Field and Master Record enter the system. Tracing each of these inward, it is at points E and F where these input data have been transformed into essential data (Valid Transaction and Valid Master Record, respectively); up to these points the raw inputs have been simply validated and edited. Points C and D are the ends of the efferent streams; at these points the outputs New Master Record and Applied Transaction Report Line, exit the system. Tracing each of these back into the system, it is at points G, H, and I where the most logical data flows first appear (Applied Transaction, Updated Master Record, and Unmatched Master Record, respectively); following these points *the data is simply being formatted.* The points E, F, G, H, and I mark the boundaries of the central transform. If these points are connected, as is shown by the dotted line, the transforms inside (5 and 6) comprise the central transform.

L24OH3
This system inputs a file name and outputs the number of words in the file. Again, identify the central transform.

b. Given a DFD, the central transform can be identified as follows:
   i Trace each physical input to the point at which the activities performed are not just editing, verifying, or otherwise cleansing the data, but are truly transforming it some way (performing calculations, using it to derive new information, etc). Mark those points. In so doing you are marking the data flow that represents the input in its most essential form.
   ii Trace each physical output backward into the system to the point where the activities are no longer simply formatting the data for output. Mark those points. In so

doing you are marking the data flow that represents the output in its most essential form.

iii Connect the points marked in steps i and ii above. The transform(s) enclosed represent the central transform.

The central transform is the part of the DFD that contains the essential functions of the system and is independent of the particular implementation of the input and output. The identification of the central transform allows the designer to clearly separate interfaces to and from the system from the essential system processing.

3. L24OH4
Once the central transform has been identified, a first-draft structure chart can be developed. The second level of the structure chart consists of a set of controller modules, one for each of the afferent streams, one for the central transform, and one for each of the efferent streams.

Consider the previous example of the system to count the number of words in a file (L24OH3). The first-draft structure chart resulting from transform analysis is shown L24OH5. Consider the cohesiveness of these modules. Read-and-validate-file-name and Format-and-display-word-count exhibit communicational cohesion. Their cohesion levels can be improved with the refinement shown in L24OH6. Now, all of the modules are functionally cohesive. This refinement represents a good preliminary design.

4. L24OH7
Structure charts produced through transform analysis are <u>balanced hierarchies</u>. The central transform is isolated from the input and output environment by placement at a separate level. The highest level modules are isolated from low-level I/O details since they see only the net results of low-level module activity.

5. The first-draft structure charts produced by transform analysis must be refined with consideration given to design criteria such as cohesion, coupling, fan-in, fan-out, and modularity. Other good examples are given in Mynatt in section 4.3. A more detailed example involving the SafeHome security system is provided in Pressman, pages 372-381.

6. While transform analysis is the most widely applied structured design technique, another method, transaction analysis, is more appropriate for "transaction centers" within a system. A transaction center occurs when a single transform in a DFD triggers multiple data streams flowing out of that activity. Transaction centers are easily recognizable.

L24OH8, L24OH9
Consider these examples as well as Mynatt's ATM example in Figure 4.12. A good way to design a transaction center is to separate it into two pieces; one to analyze the transaction (the afferent to the transaction center), and one to dispatch the transaction. This separates the different transactions at a very high level and discourages the tendency to share common elements.

Discuss the types of structure charts that result. For example, applying transaction analysis to the DFD in L24OH9 yields the first-draft structure chart in L24OH10. Other good examples are given in Mynatt in section 4.4. A more detailed example involving the SafeHome security system is provided in Pressman, pages 382-389.

7. L24OH11
The structure charts produced by transform analysis and transaction analysis must be refined with consideration given to design criteria such as cohesion, coupling, fan-in, fan-out, and modularity. They must also be reviewed to verify that the final structure chart meets the requirements represented by the DFDs. Discuss the concordance example in Mynatt, Section 4.3.4, pages 162-167.

PROCEDURE:
<u>teaching method and media</u>:

<u>vocabulary introduced</u>:
transform analysis
afferent streams
efferent streams
central transform
factoring
balanced hierarchy
transaction analysis
transaction center

<u>INSTRUCTIONAL MATERIALS</u>:
<u>overheads</u>:

L24OH1	Transform flow
L24OH2	DFD for master file update
L24OH3	DFD for word counter
L24OH4	First-level factoring
L24OH5	First-level structure chart of word counter
L24OH6	Refinement of word counter structure chart
L24OH7	Balanced hierarchies
L24OH8	Transaction center
L24OH9	Ship control system
L24OH10	Ship control system
L24OH11	Refinement and verification of structure chart

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

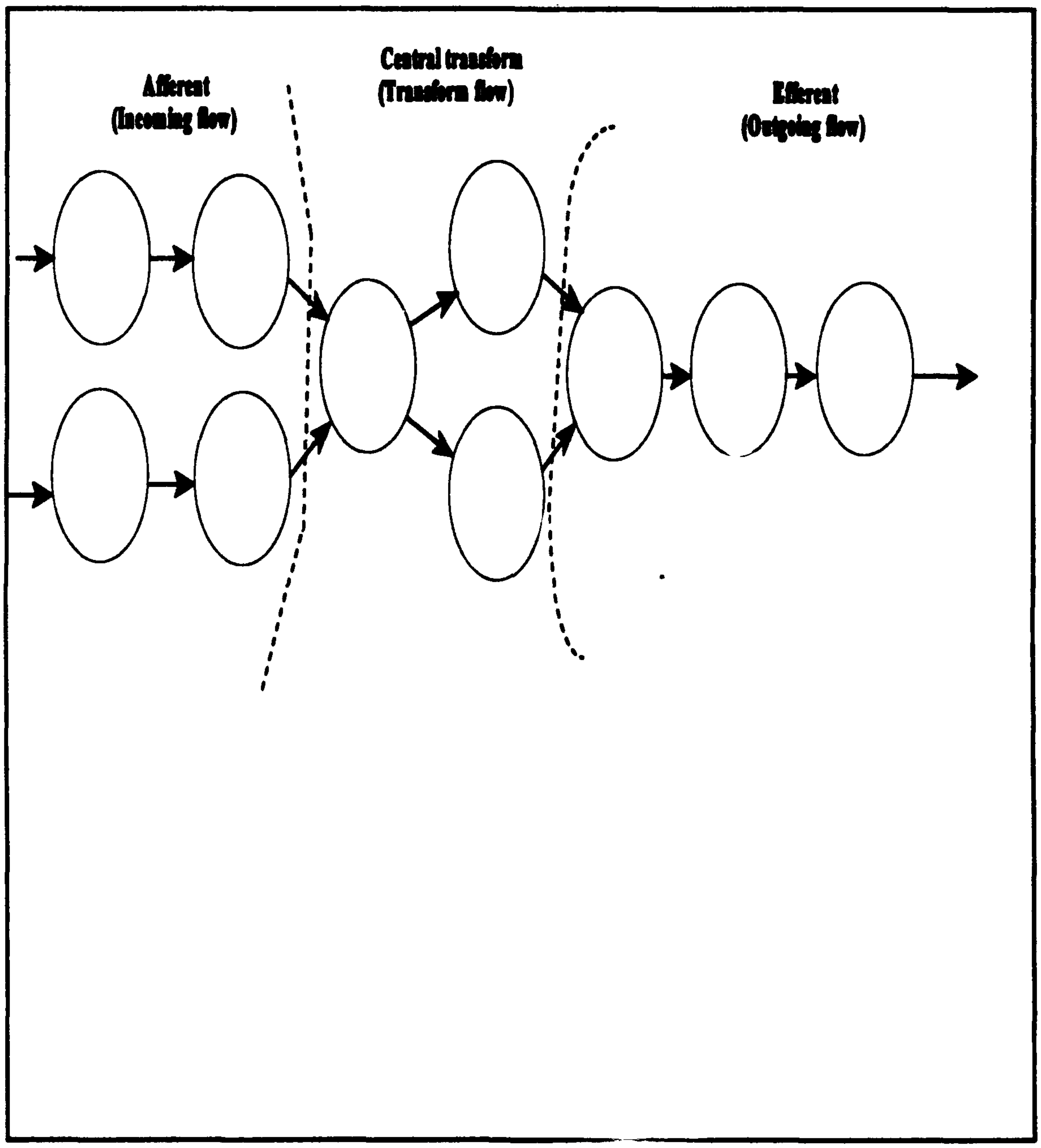
READING ASSIGNMENTS:

Sommerville Chapter 2 (pp. 222-228)
Mynatt Chapter 4 (pp. 143-169)

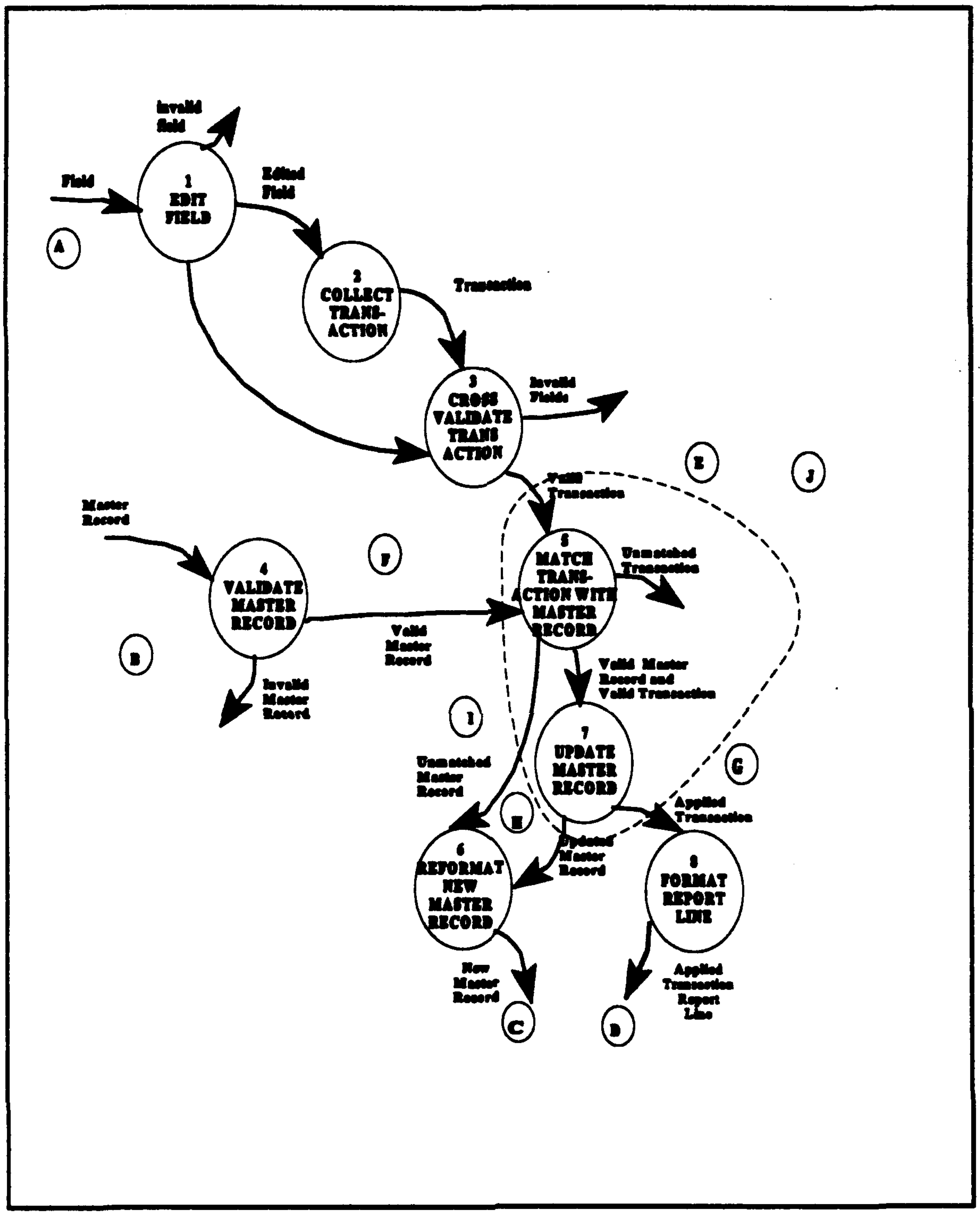
RELATED READINGS:

Ghezzi Chapter 7 (pp. 394-402)
Pressman Chapter 11 (pp. 369-389)
Schach Chapter 10 (pp. 299-302)

Transform Flow

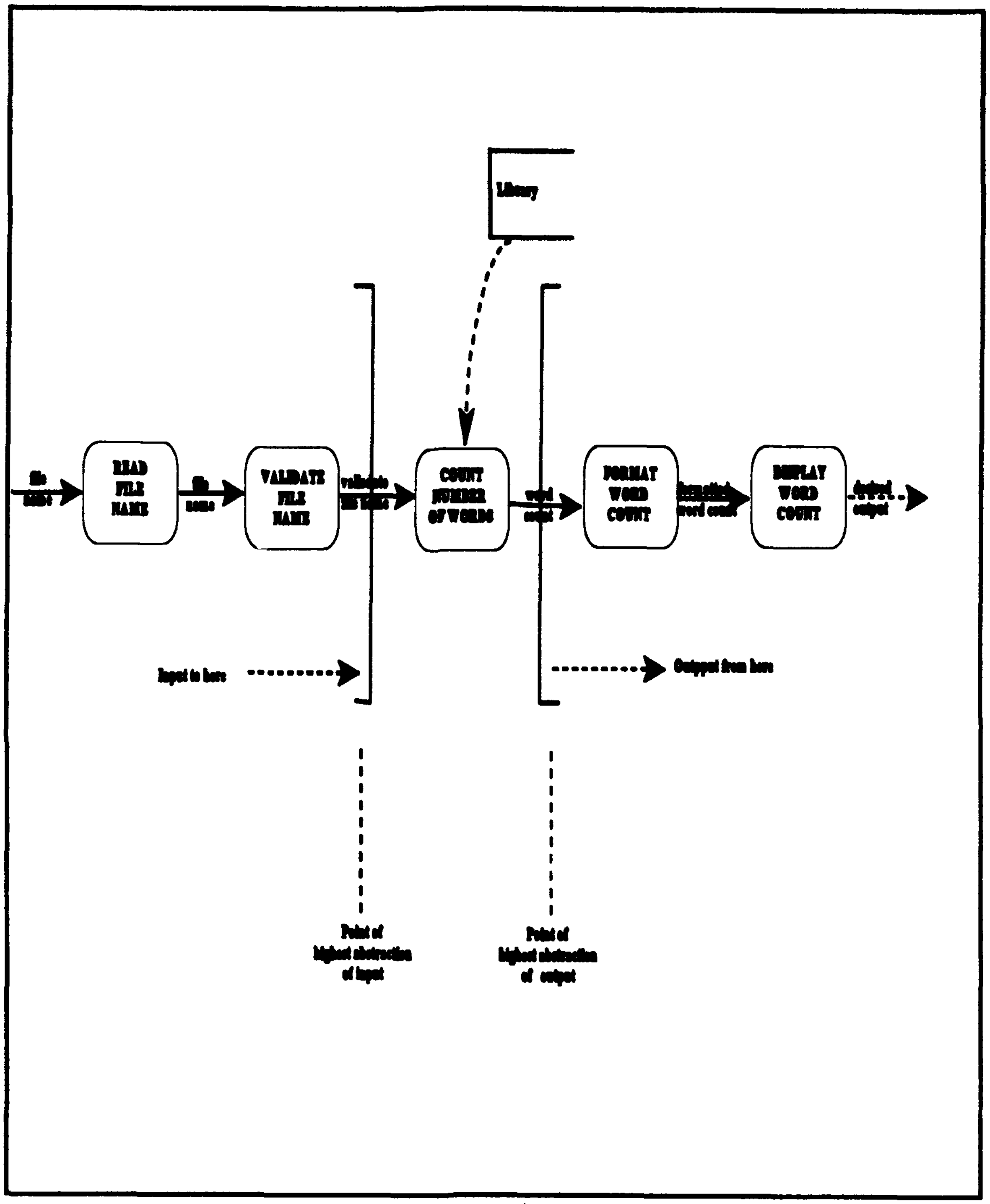


DFD For Master File Update



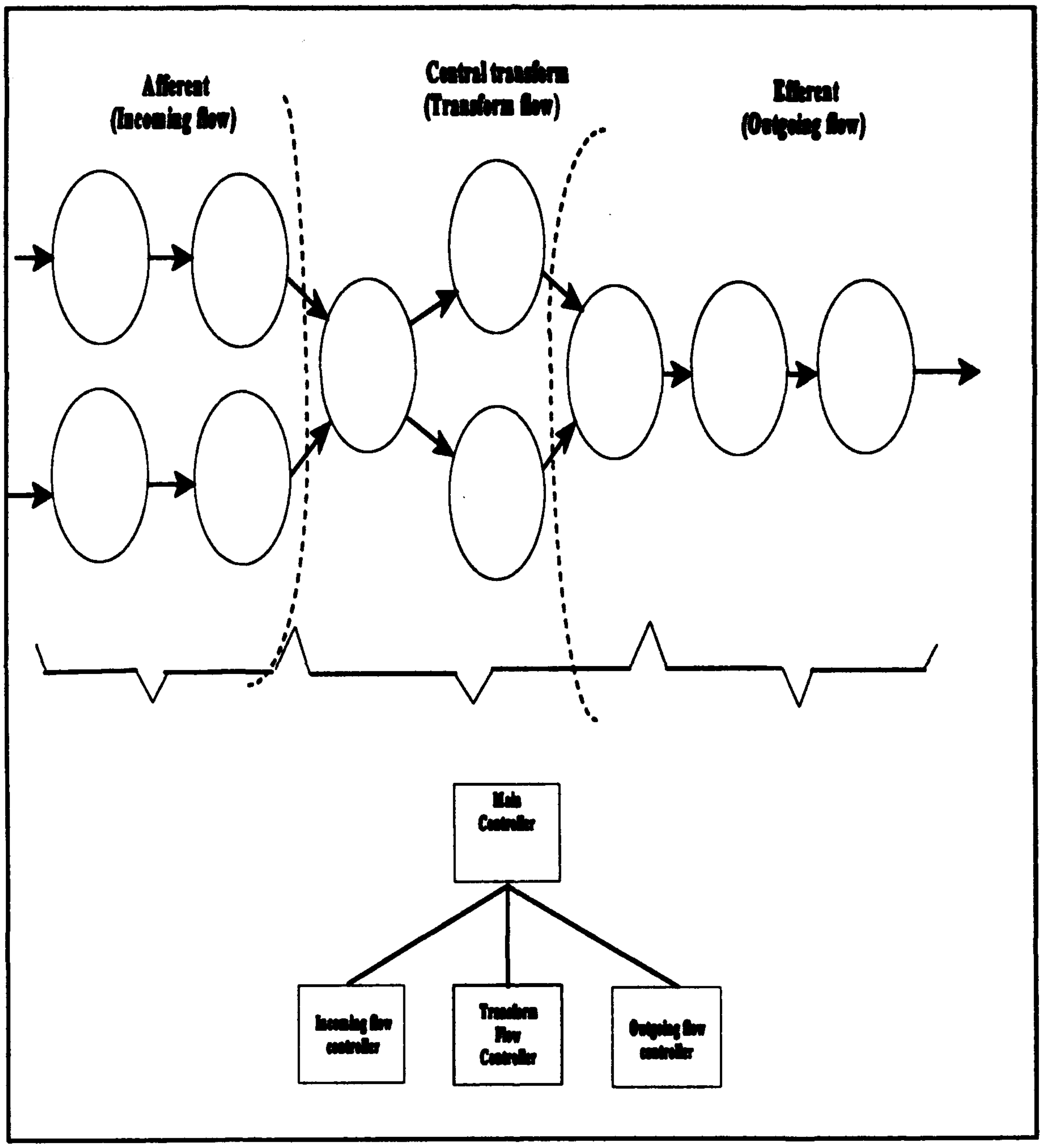
Yourdon Seminar Notes

DFD For Word Counter



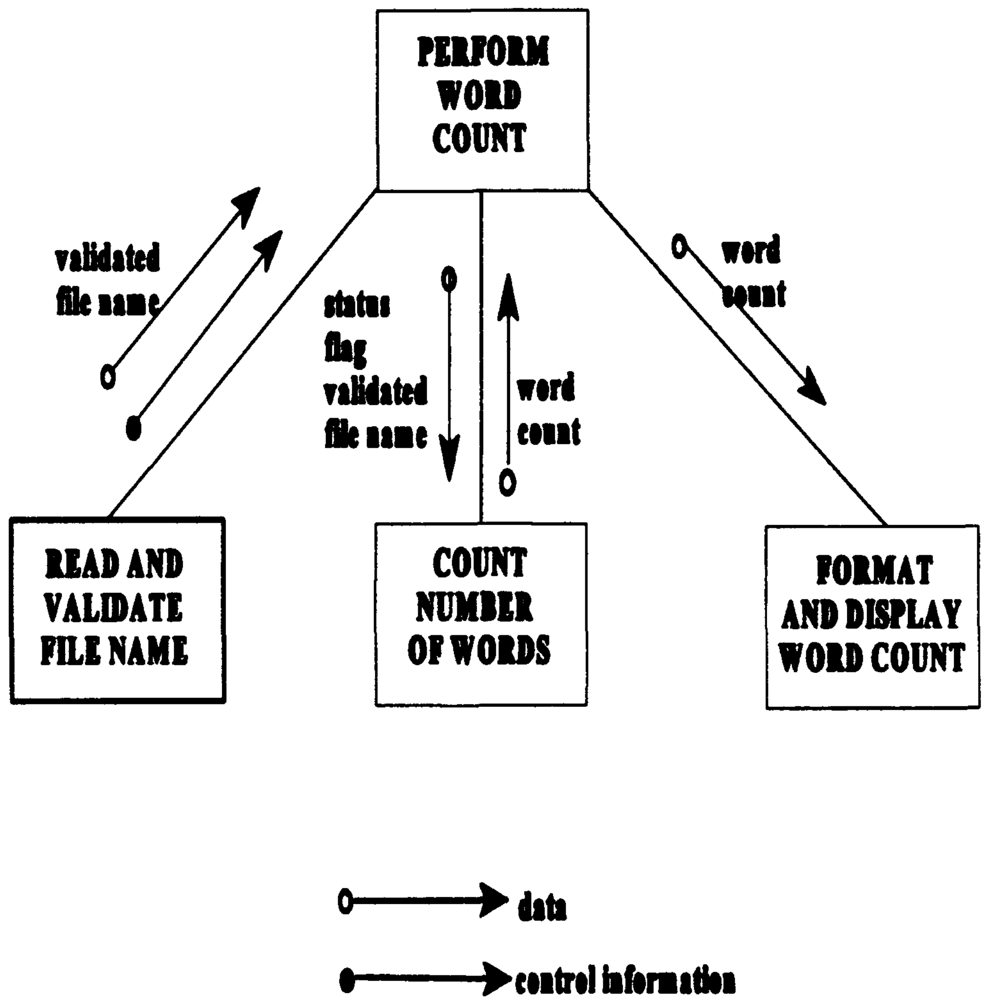
Schach, Figure 10.3

First Level Factoring



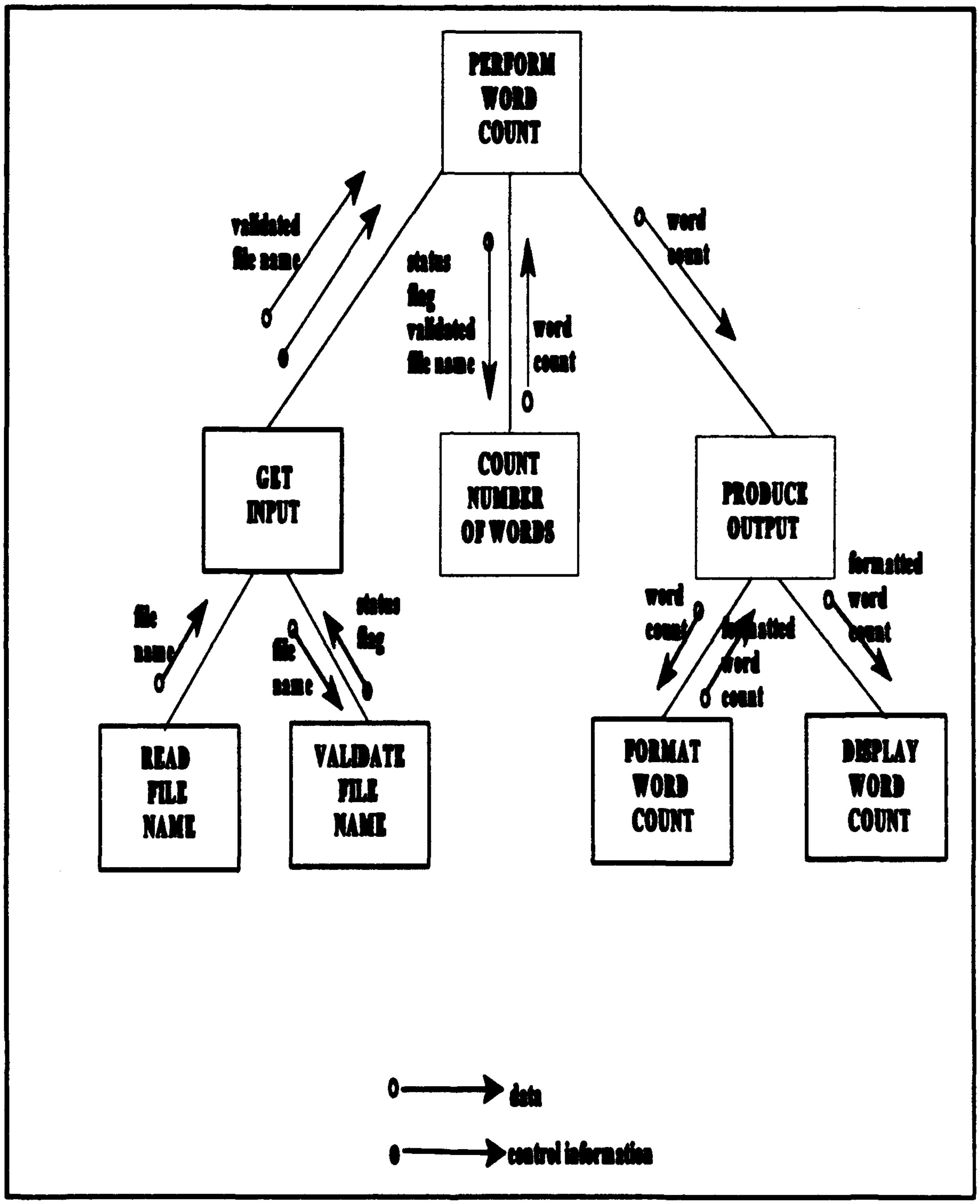
Pressman, Figure 11.9

First-Level Structure Chart of Word Counter



Schach, Figure 10.4

Refinement of Word Counter Structure Chart



Schach, Figure 10.5

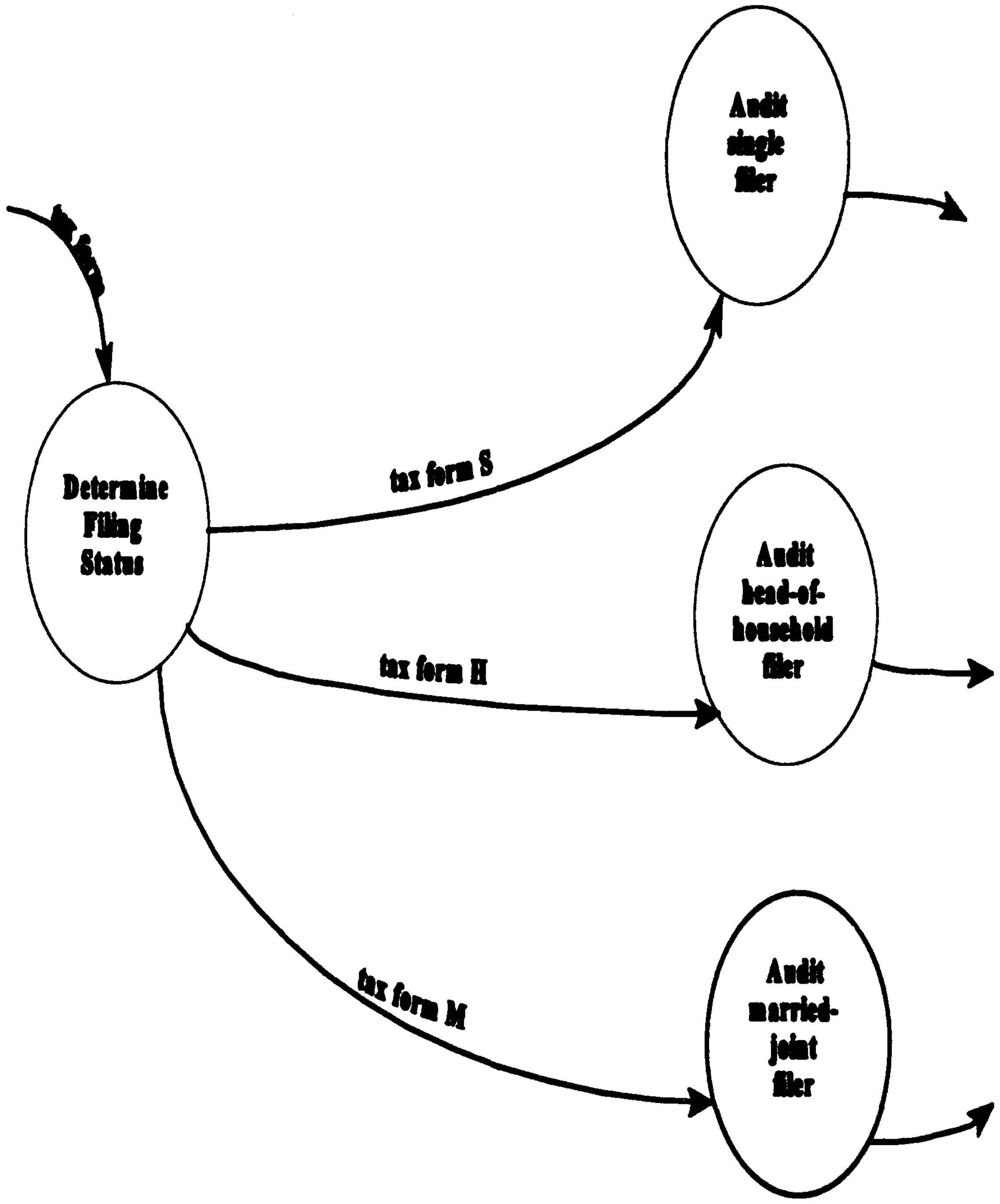
Balanced Hierarchies

Structure charts produced through transform analysis are balanced hierarchies.

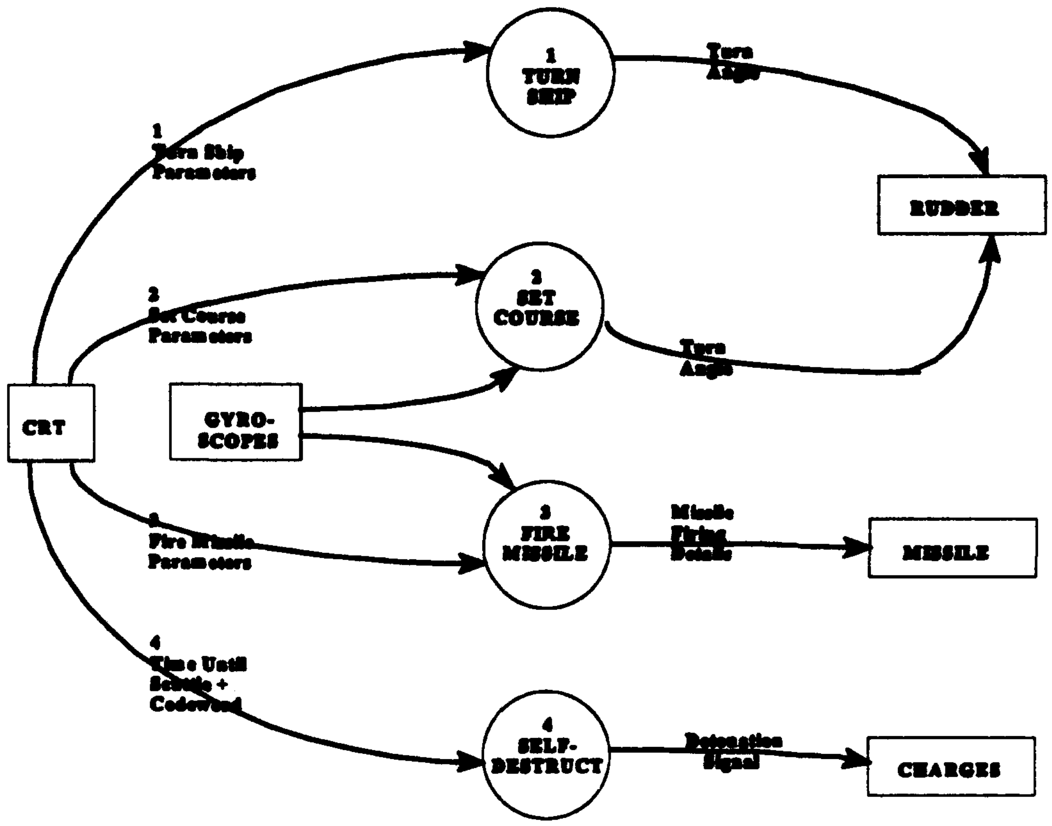
The central transform is isolated from the input and output environment by placement at a separate level.

The highest level modules are isolated from low-level I/O details since they see only the net results of low-level module activity.

Transaction Center Example



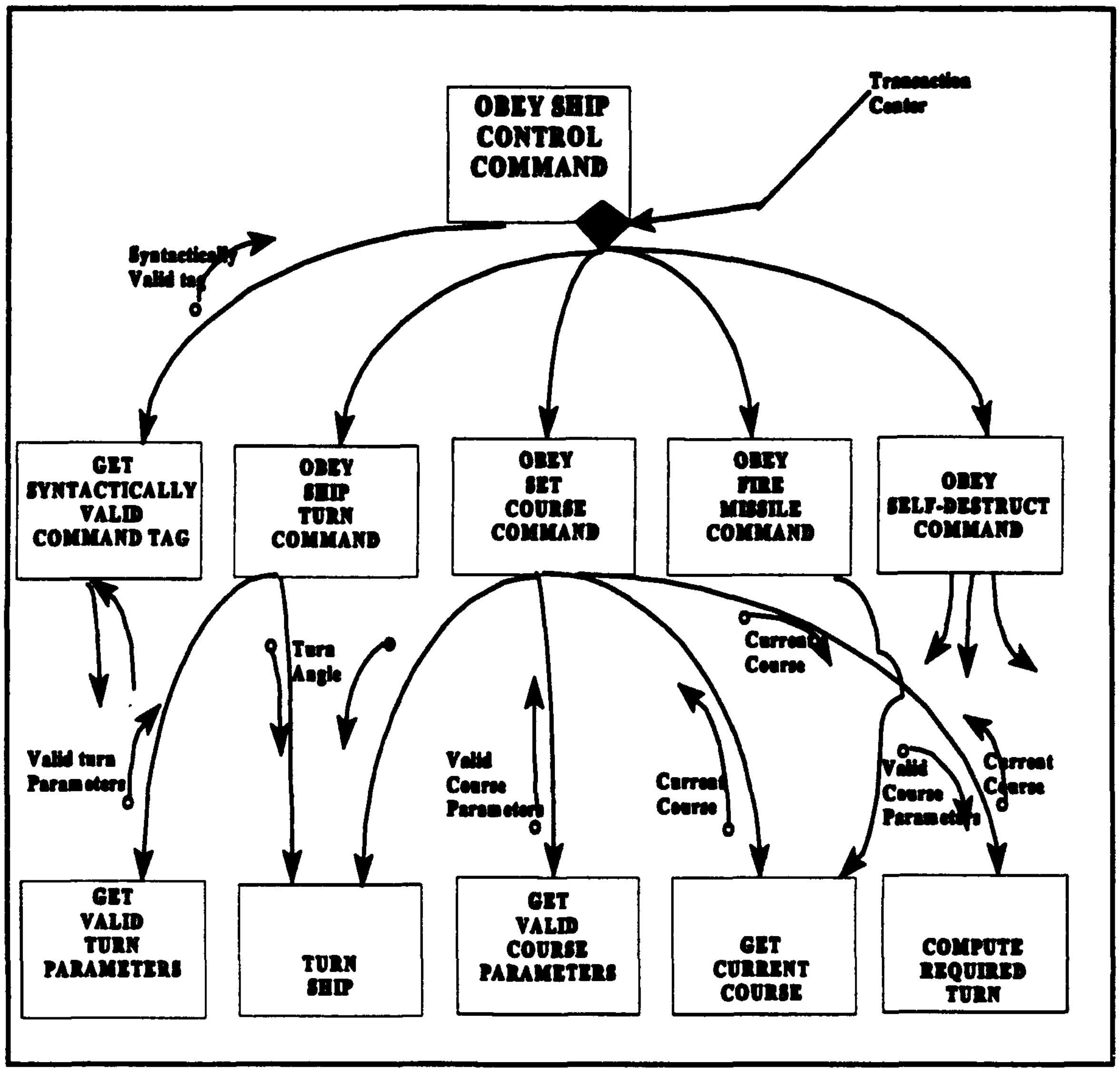
Ship Control System



Transaction Tag	Transaction Type	Transaction Effect
Turn	Turn ship.	Turn ship from present angle by specified amount.
Set	Set ship course.	Set ship to absolute course.
Fire	Fire missile.	Fire missile in specified direction.
Scuttle	Self-destruct.	Blow up ship after specified time.

Yourdon Seminar Notes

Ship Control System Structure Chart



Yourdon Seminar Notes

Refinement and Verification Of Structure Chart

The structure charts produced by transform analysis and transaction analysis must be refined with consideration given to design criteria including modularity, cohesion, coupling, fan-in, and fan-out.

The refined structure charts must also be reviewed to verify that they meet the requirements.

LECTURE NUMBER: 025

TOPIC(S) FOR LECTURE:
Coupling and cohesion

INSTRUCTIONAL OBJECTIVE(S):

1. Understand the design goals for cohesion.
2. Understand and distinguish between the various cohesion levels.
3. Understand the design goals for coupling.
4. Understand and distinguish between the various coupling levels.
5. Evaluate a design based on its coupling and cohesion characteristics.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

In earlier lectures we have discussed coupling and cohesion as design criteria. Recall that cohesion is a measure of internal strength of a component; a measure of how well the internal elements of a component work towards the goal of the module. Thus, in design we want to maximize cohesion; to design highly cohesive components. Recall also that coupling is a measure of the dependencies between components; a measure of the relationship between components. Thus, in design we want to minimize coupling; to restrict the dependencies between components to those that are necessary.

(Learning Label- Today we are going to learn ...)
Today we are going to discuss various levels of coupling and cohesion and how to evaluate a design based on its coupling and cohesion.

CONTENTS:

L25OH1

1. One attribute of a design is its modularity. What do we mean by that; exactly what is a module? Consider a module as a black box with four basic attributes.
   - i It interface, Input and output - what it gets from its invoker (input) and what it returns (output)
   - ii Function - what it does to its input to produce its output
   - iii Mechanics - procedural logic to performs its function
   - iv Internal data - its own private data or work-space or data structure

   Items i and ii comprise the external view of a module. Items iii and iv comprise the internal view. Preliminary design is concerned with the external view. Functional independence is a key to good design, and thus, to software quality. Why? Because the design will be maintainable, testable, and have a higher potential for reuse. Ideally, this is achieved by developing modules that are single-minded (do a single, clearly-defined function), avoid interactions with other modules except where necessary. In those cases, where interaction is necessary, keeps it as simple as possible.

2. Cohesion is a design criteria; a measure by which we can evaluate a design. Cohesion is a measure of a module's internal strength; a measure of how well a module's internal elements are related to each other. Ideally, a cohesive module does just one thing. A function with no side-effects is an excellent example of a cohesive module. A number of levels of cohesion have been identified. L25OH2 As the overhead indicates, these levels represent a spectrum. The scale in <u>not</u> linear. The low end (coincidental) is very bad, and should always be avoided,whereas the middle levels, which are not that far away from the high end (functional), are sometimes unavoidable. L25OH3 The lower levels of cohesion lead to a maintenance problem.

   a. Coincidental cohesion - there is little or no meaningful relationship between the elements of the module. Such modules perform multiple, unrelated tasks. This is the worst type of coupling and also easiest to avoid. The problems are obvious: difficult to maintain and offer little opportunity for reuse.

   b. Logical cohesion - module performs a series of related actions, one of which is selected by the calling module. This occurs when elements are grouped into a class of related functions and placed into a single module. Examples are a module that handles all output, general purpose error handling, and modules that perform all input. Problems with logical cohesion include a complex interface , hard to understand module, and code for different actions may intertwine, causing maintenance problems. For example in L25OH4, the addition or deletion of hardware would cause significant modification in the module.

   c. Temporal cohesion - module performs a series of actions that are related by time, actions that must be done at the same time (or in the same time span). Typical examples are initialization modules that do a variety of things (like open files, clear counters, initialize flags); or "wrap-up" modules that close files, compute final totals and averages, and print final report. CS1 instructors tend to emphasize these types of modules; (e.g., initialize all conditions and totals, other housekeeping chores). This leads to tight coupling.

   d. Procedural cohesion - module performs a series of actions that must be done in a particular order; the elements are related more to program procedure than to program function. They often tend to cut across functional boundaries. Not always bad, in fact from this point upward on the cohesion spectrum, the levels of cohesion are significantly more maintainable than the lower levels. An example of procedural cohesion is a module that reads a part number from a data store and updates a repair record in a maintenance data store. Although procedurally cohesive modules are more maintainable than those with lower levels of cohesion, they are not easily reusable.

e. Communicational cohesion and sequential cohesion - perform a series of actions related by a sequence of steps; an assembly-line order. If all the actions are preformed on the same data structure than the cohesion is communicational.

   Examples: Determine length and slope of line
   Read record and eliminate duplicates.
   Format and verify voter profile.

   Tasks at this level are directly related to the problem so maintainability is not bad but, again, decreases potential for reuse

f. Functional cohesion - module performs a single task and each part of the modules is necessary to perform that task.

   Examples: Calculate sales commission
   Get temperature of furnace
   Determine students GPA

   Note: the cohesiveness of many of the earlier examples could be improved to functional by breaking them into multiple modules.

   Discuss why such modules are easier to maintain
   - Fault isolation
   - Easy to understand
   - Less chance of impacting other modules
   - Easier to extend/replace
   - Better chance of reuse

   Example: Discuss this example with the students. L25OH5

g. Informational cohesion - This is an additional level of cohesion identified by Schach. A component exhibits informational cohesion if it has a number of elements, each preforming an action on <u>the same data structure</u>, and, <u>each element has its own entry point</u>.

Example: L25OH6 The difference between this and logical cohesion is that here the various elements, each performing one action, are independent whereas in logical cohesion the elements are intertwined. This lies just below functional on the cohesion scale.

Summary on cohesion: If a module exhibits more than one type of cohesion, it is labeled as the worst of those types. One should develop modules that have a single problem-related function. This increases independence, clarity, maintainability, and reuse. A functionally cohesive module can be accurately described with a simple sentence containing an imperative verb and a specific singular object. Otherwise, the module is less than functionally cohesive. L25OH7, L25OH8 is Meilor Page-Jones' organization of cohesion and its trade-offs. The Y-axis represents the lifetime cost per amount of functionality provided and the X-axis is the levels of cohesion. Page-Jones has an excellent discussion of these trade-offs.

3. Coupling is another design criteria; a measure by which we can evaluate a design. Coupling is:
   i a measure of the dependence between two modules
   ii a measure of interconnection between module
   ii it is the degree of interaction between two modules.

   A major design goal is to minimize dependencies by; developing loosely coupled modules. Low coupling is achieved by eliminating unnecessary relationships and minimizing the number and "tightness" of necessary relationships. A number of levels of coupling have been identified. L25OH9 spectrum (p 336 Pressman). As the overhead indicates, these levels represent a spectrum. As with the cohesion spectrum, the scale in not linear.

   L25OH10

   a. Content coupling - One module refers to the internals of another.

      Examples (Assume modules A and B) include:
         A modifies a statement of B
         A refers to local data of B
         A branches inside B

      These are c easy to avoid, inexcusable, violations of anybody's programming standards.

   o. Common coupling - Two modules have access to same global data area.

c. Control coupling - One module passes an element of control to another module, i.e. explicitly controls its logic. Examples of control are function codes, flags, and switches. If a table lookup routine passes back a flag "entry not found" there is no control coupling. If however, it passes back "entry not found, add this item" then the table look up module that called it is control coupled. In the latter case the boss is being told what to do; in the former case the boss is being apprised of the situation.

d. Stamp coupling - One module passes a data structure as a parameter but the called module operates on some but not all of the individual components of the data structure. Stamp coupling exposes modules to more data than they need. This exposes the data structure to corruption.

e. Data coupling occurs when parameters are passed or a data structure, all of whose elements are needed by the called module. The fact that a data structure is passed does not entail stamp coupling.

f. Two modules may exhibit more than one type of coupling. In such cases, the degree of coupling is considered as the worst of the types exhibited.

L25OH11 is an example of different couplings. Discuss this with the class and go over the answers L25OH12.

g. Considerations in designing modules:
   i imagine modules as library functions; how would they be easiest to understand; how might they be reusable?
   ii Assume each module will be implemented by a different programmer. How independently can the programmers work? Are there any assumptions, constraints, or conventions that one module need be aware of and how likely are they to change? Can the change be isolated to a single module?

If two modules are highly coupled, there is a higher probability that a change in one will require a change to the other. L25OH13 is Page-Jones' organization of coupling and its tradeoffs.

PROCEDURE:

teaching method and media:

vocabulary introduced:

cohesion
coincidental cohesion
logical cohesion
temporal cohesion
procedural cohesion
communicational cohesion
sequential cohesion
functional cohesion
informational cohesion
coupling
content coupling
common coupling
control coupling
stamp coupling
data coupling

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>:

L25OH1	Attributes of a module
L25OH2	Cohesion Spectrum (Pressman, p 334)
L25OH3	Cohesion level definitions
L25OH4	Logical cohesion example (Schach)
L25OH5	Structure chart showing cohesion of each module (Schach, Fig 9.7)
L25OH6	Module with informational cohesion (Schach, Fig 9.6)
L25OH7	Guidelines for determining cohesion level (Pressman, p. 335)
L25OH8	Costs as function of cohesion level (Page-Jones, Table 6.2, Fig 6.15)
L25OH9	Coupling spectrum (Pressman, p 336)
L25OH10	Coupling level definitions
L25OH11	Coupling example (Schach, Fig 9.11 - 9.12)
L25OH12	Coupling example (Schach, Fig 9.13)
L25OH13	Qualities of coupling levels (Page-Jones, Table 5.2)

<u>handouts</u>:

<u>RELATED LEARNING ACTIVITIES</u>:
(labs and exercises)

<u>READING ASSIGNMENTS</u>:
Mynatt Chapter 4 (pp. 144-150)

<u>RELATED READINGS</u>:
Ghezzi Chapter 3 (pp. 49-52)
Pressman Chapter 10 (pp. 332-337)
Schach Chapter 9 (pp. 235-253)

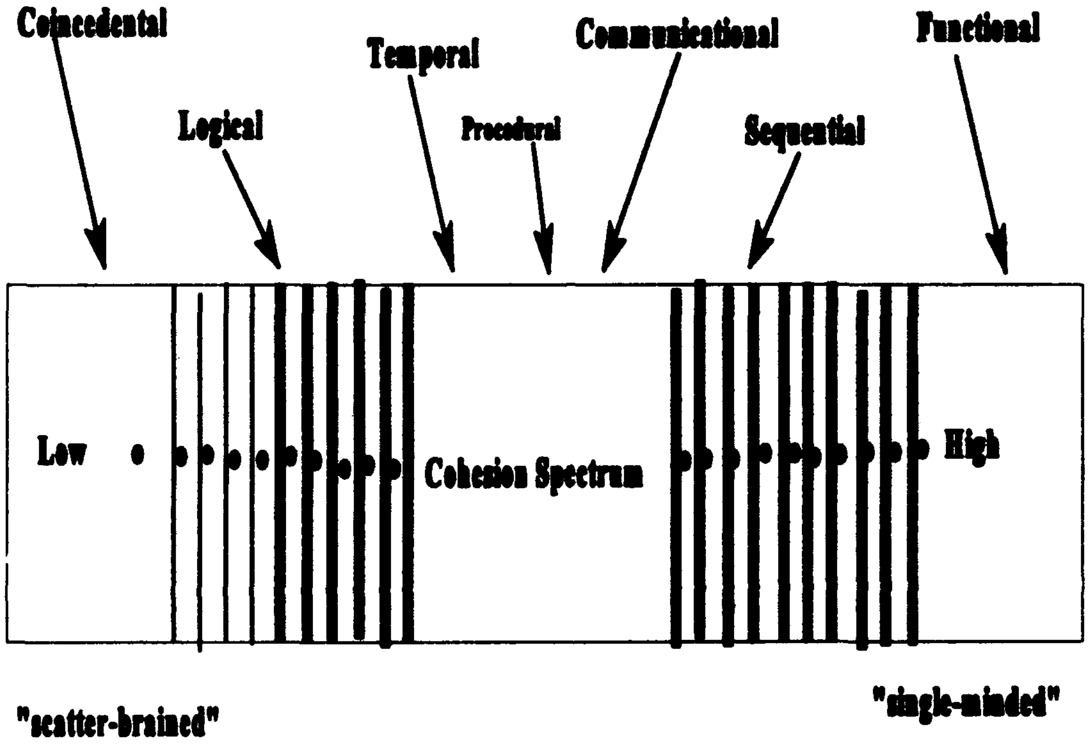
Melir Page-Jones, Practical Guide to Structured Design, 2nd ed, 1988,

Attributes of a Module

Interface -	Its input and output
Function -	What it does to its input to transform it into output
Mechanics -	Internal logic (code)
Internal data	

Cohesion

A measure of the relative functional strength
of a module



Pressman 334

COHESION LEVELS

Coincidental -	Little or no meaningful relationship between elements of the module
Logical -	Performs series of logically related functions in module; functions falling into some general category
Temporal -	Performs series of actions related by time (that must all be done at same time or in same time span)
Procedural -	Performs series of actions that must be done in a particular order; elements related more to procedure than to function
Communicational Sequential-	Performs series of actions related by sequence (output of step is input to next) or all of the actions performed on the same data
Functional -	Performs a single task and each element of module is necessary to perform that task

Example of Logical Cohesion

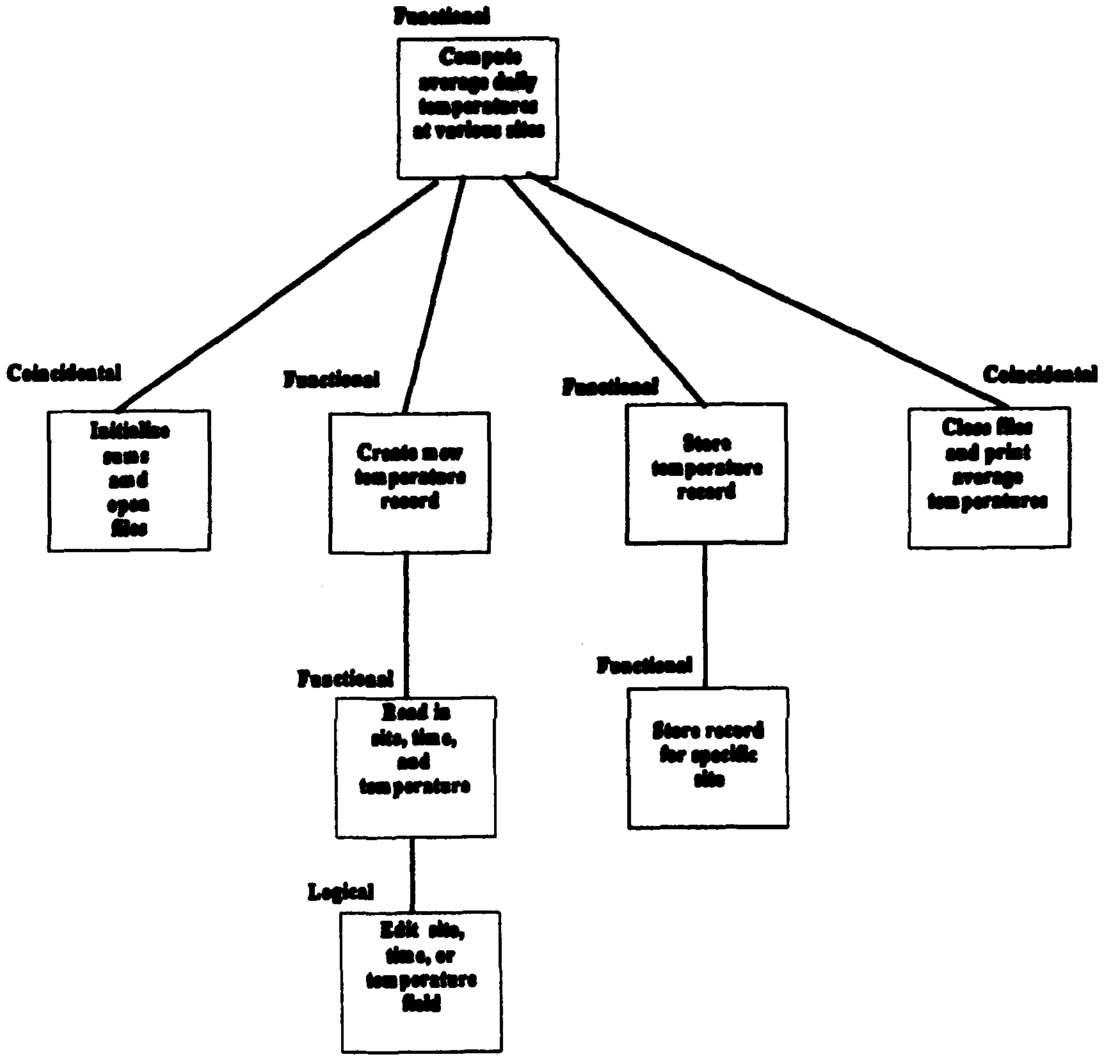
Module Performing All Input and Output
1. Code for all input and output
2. Code for input only
3. Code for output only
4. Code for disk and tape I/O
5. Code for disk I/O
6. Code for tape I/O
7. Code for disk input
8. Code for disk output
9. Code for tape input
10. Code for tape output

. . . . .
. . . . .
. . . . .

37. Code for keyboard output

Source : Schach, Fig 9.5

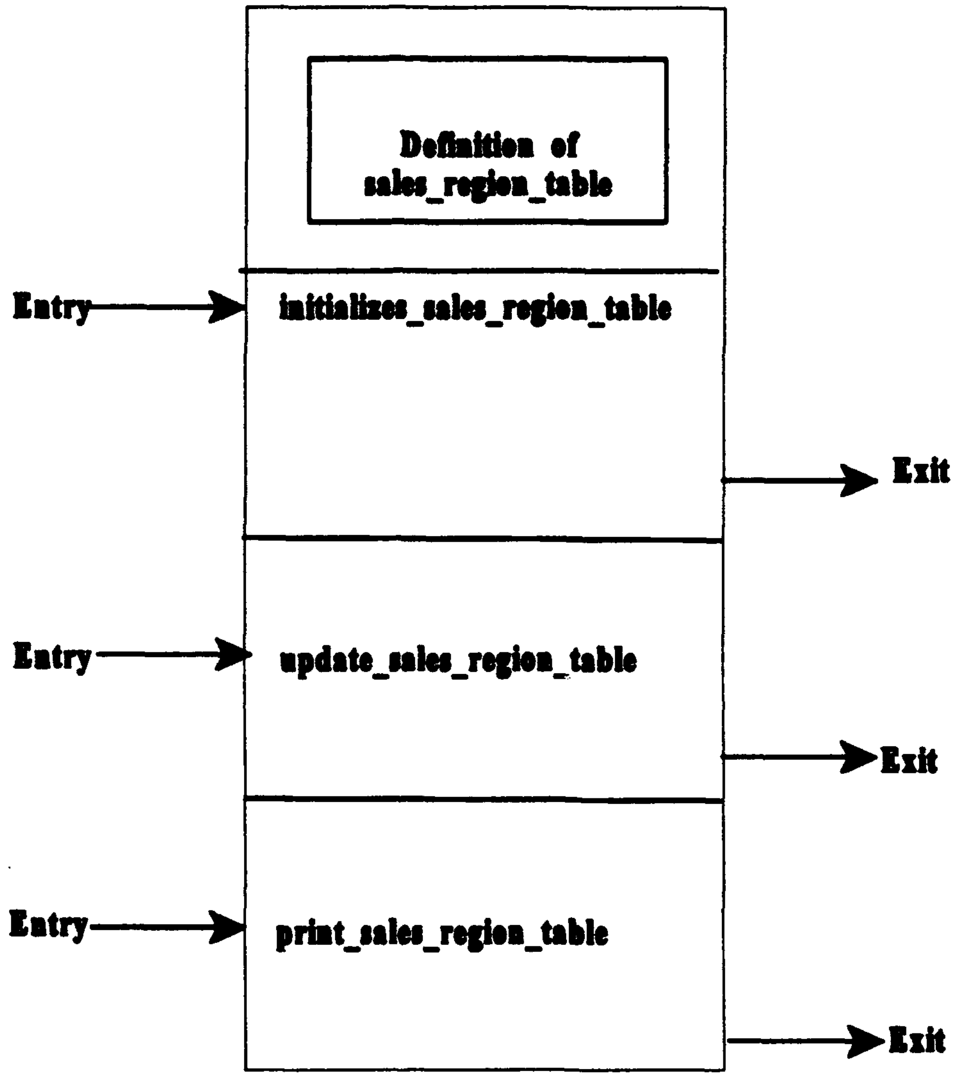
Module Interconnection Diagram Showing Cohesion of Each Module



Functional
Compute average daily temperatures at various sites
Coincidental
Initialize sums and open files
Functional
Create new temperature record
Functional
Store temperature record
Coincidental
Close files and print average temperatures
Functional
Read in site, time, and temperature
Functional
Store record for specific site
Logical
Edit site, time, or temperature field

Schach, Figure 9.7

Module with Informational Cohesion



Guidelines for Determining Cohesion Level

A useful technique in determining whether a module is functionally bound is writing a sentence describing the function (purpose) of the module, and then examining the sentence. The following tests can be made:

1. If the sentence has to be a compound sentence, contains a comma, or contains more than one verb, the module is probably performing more than one function; therefore, it probably has sequential or communicational binding.

2. If the sentence contains words relating to time, such as "first", "next", "then", "after", "when", "start", etc., then the module probably has sequential or temporal binding.

3. If the predicate of the sentence doesn't contain a single specific object following the verb, the module is probably logically bound. For example, *Edit All Data* has logical binding: *Edit Source Statement* may have functional binding.

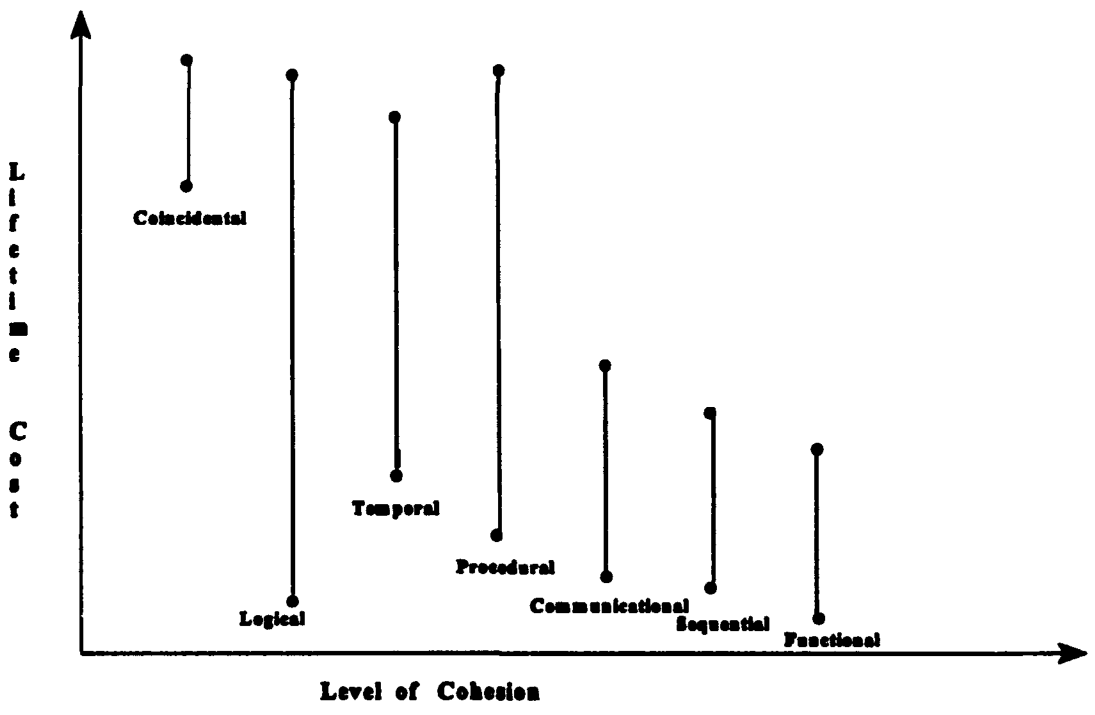
4. Words such as "initialize", "clean-up", etc., imply temporal binding. Functionally bound

modules can always be described by way of their elements using a compound sentence. But if the above language is unavoidable while still completely describing the module's function, then the module is probably not functionally bound.

Source: Pressman, p 335

Costs As Function of Cohesion Level

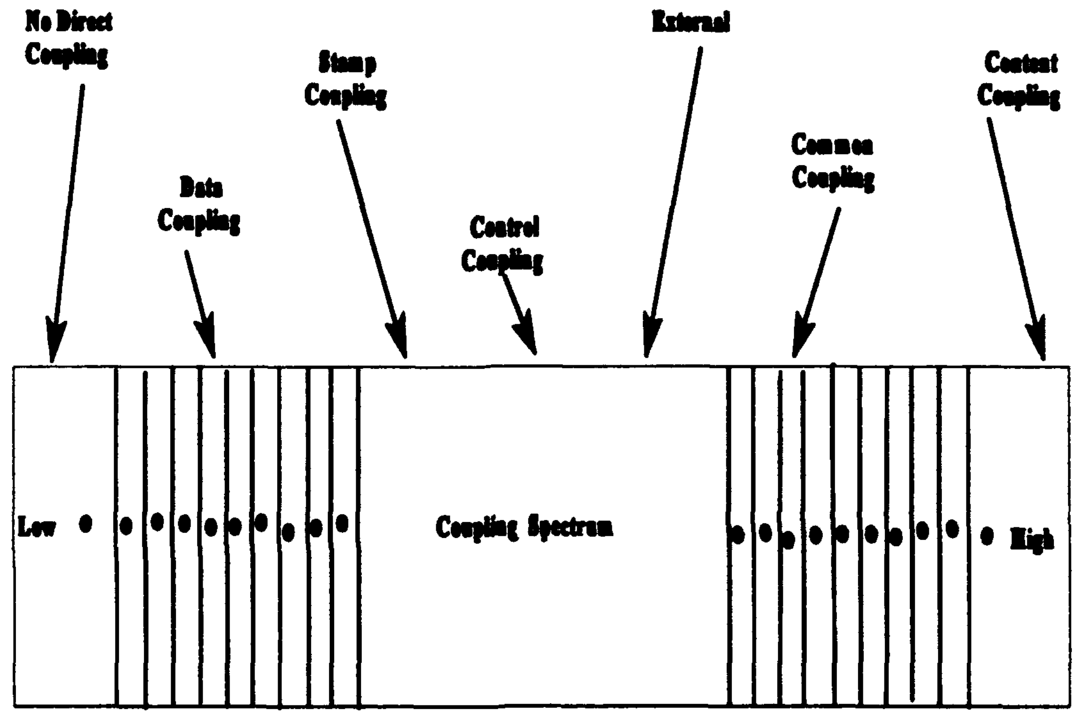
Cohesion level	Coupling	Cleanliness of implementation	Modifiability	Understandability	Effect on overall system maintainability
Functional	Good	Good	Good	Good	Good
Sequential	Good	Good	Good	Good	Fairly good
Communicational	Medium	Medium	Medium	Medium	Medium
Procedural	Variable	Poor	Variable	Variable	Bad
Temporal	Poor	Medium	Medium	Medium	Bad
Logical	Bad	Bad	Bad	Poor	Bad
Coincidental	Bad	Poor	Bad	Bad	Bad



Source: Page-Jones

Coupling Spectrum

A Measure of the Interdependence Among Software Modules

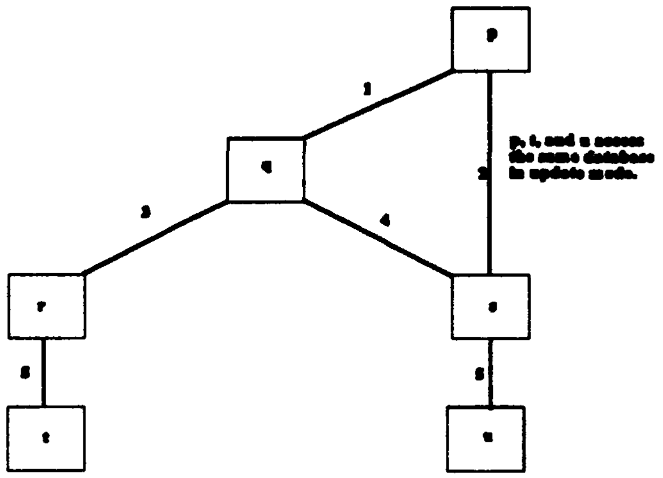


Pressman, Figure 10.12

COUPLING LEVELS

Content-	One refers to internals of other
Common -	Have access to same global data area
Control -	Communicate at least one element of control
Stamp -	Communicates a data structure and the called module operates on some <u>but not all</u> of the individual components of the data structure
Data -	Only parameters communicated or a data structure in which all of the elements are needed by the called module

Coupling Examples (Schach) Module Interconnection Diagram for Coupling Example



**Interfa
ce Description**

Number	In	Out
1	aircraft type	status flag
2	----	list of aircraft parts
3	function code	----
4	----	list of aircraft parts
5	part number	part manufacturer
6	part number	part name

Coupling Between Pairs of Modules

a. Are we building the right product?

Validation involves checking that the program as implemented meets the expectations of the customer as specified in the requirements specification documentation. In other words, we want to demonstrate through tests whether or not the software product meets the customer's expectations. The completed end product is tested.

c. Dynamic analysis is the primary technique used in accomplishing validation.

4. Discuss verification and validation activities of each phase of the traditional process model are examined. L21OH5, L21OH6 L21OH7 -Review of software products is traced back to its requirements. L21OH8, L21OH9 - Maintenance is a reiteration of the software development life cycle. Maintenance also re-applies previous test cases to assure no loss of functionality during maintenance changes.

5. Acceptance criteria are important in the accomplishment of validation. Without established, agreed upon criteria by which to judge validation, certification of customer satisfaction is difficult. The role of acceptance criteria within the software requirements specification is discussed. The acceptance tests provide predefined criteria for functionality, performance, interface quality, and other identified quality attributes.

L21OH10

6. The acceptance criteria described above are often called explicit requirements because the customer has explicitly stated them in the software requirements specification. There is another type of requirements which developers need to be concerned about; these requirements are called implicit requirements. Implicit requirements are quality factors which the customer desires or expects in the software but may not explicitly state. Examples of implicit requirements include reliability and robustness.

L21OH11

7. The activities of verification and validation should be thoroughly planned and documented in a Software Verification and Validation Plan. This document serves as the blueprint for all V&V activities. Give examples of the major bullets on this overhead.

PROCEDURE:

teaching method and media:

vocabulary introduced:

verification
validation
acceptance criteria
static analysis
dynamic analysis
formal analysis

INSTRUCTIONAL MATERIALS:

overheads:

L21OH1	Verification and Validation
L21OH2	Types of Analysis Used in Verification & Validation
L21OH3	Verification
L21OH4	Validation
L21OH5	Requirements Analysis and Definition Phase
L21OH6	Design Phase
L21OH7	Implementation Phase
L21OH8	Testing Phase
L21OH9	Maintenance Phase
L21OH10	Framework for a Software Requirements Specification
L21OH11	Software Verification and Validation Plan

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 018 - Preliminary requirements presentation/review
Preliminary users manual presentation and review

READING ASSIGNMENTS:

Sommerville Chapter 19 (pp. 373-386)
Sommerville Chapter 22 (pp. 425-439)

RELATED READINGS:

Ghezzi Chapter 6 (pp. 255-344)
Pressman Chapter 19 (pp. 632-663)

Verification and Validation (V&V)

Major approaches within software engineering process models for ensuring the production of quality software

Complementary but distinct

Continuing process through each stage of software life cycle

Two objectives:

1. Discovery of defects in any development product
2. Assessment of whether or not the system satisfies specified requirements

Types of analysis:

Static analysis

Dynamic analysis

Formal analysis

L21OH1

Types of Analysis Used in V&V

Static Analysis
- No execution involved
- Manual or automated examination

- examples:
  - Software reviews
  - Static program analyzers

Dynamic Analysis
- Execution involved

- Examines functional, structural, or computational aspects of software
- examples:
  - Unit testing
  - integration testing
  - Acceptance testing

Formal Analysis
- Use of mathematical techniques to evaluate product
- examples:
  - Symbolic execution
  - Proof of correctness

Verification

Are we building the product right?

Evaluate the end product of each phase

Look for errors generated within a phase and/or by the transformation between phases

Tasks are to assume that the products of each software life cycle phase:

1. Comply with previous life cycle phase requirements and products

2. Satisfy the standards, practices, and conventions of the phase

3. Establish the proper basis for initiating the next life cycle phase activities

Validation

Are we building the right product?

Checking that the system as implemented meets the expectations of the software procurer/customer

Tasks are to validate that the end product complies with established software and system requirements

Requirements Analysis and Definition Phase

Verification activities:
- Formal review of requirements specification document
- Review of project plan document
- Review of preliminary user manual

Validation activities:
- Delineation of acceptance criteria
- Generation of requirements-based test cases

Development of Project V&V Plan

Design Phase

Verification activities:
- Formal review of design documents
- Review of Project V&V Plan
- Generation of test plans for unit, design-unit, and system testing
- Generation of design-based test cases
- Review of test plans

Validation activities:
- Generation of test plan for acceptance testing
- develop a requirements traceability matrix

Completion of Project V&V Plan

Implementation Phase

Verification activities:

- Review of software products
- Unit testing
- Generation of code-based test cases

Validation activities:

- Develop a component to design traceability matrix

Testing Phase

Verification activities:
- Generation of code-based test cases
- Design unit testing
- System testing

Validation activities:
- System testing
- Acceptance testing

Maintenance Phase

Verification activities:
- All previous activities

Validation activities:
- All previous activities
- Regression testing
- Generation of test cases for validating modifications

Framework for Software Requirements Specification

1 Introduction
  1.1 System reference
  1.2 Business objectives
  1.3 Software project constraints
2 Software description
  2.1 Objects and operations
  2.2 Flow model
  2.3 Data dictionary
  2.4 System interface dictionary
3 Processing narratives
  3.n Transform n description
    3.n.1 Processing narrative
    3.n.2 Restrictions/limitations
    3.n.3 Performance requirements
    3.n.4 Design constraints
    3.n.5 Supporting diagrams
4 Validation/acceptance criteria
  4.1 Testing strategy
  4.2 Classes of tests
  4.3 Expected software response
  4.4 Special considerations
5 Bibliography
6 Appendix

Software Verification and Validation Plan

Plan for the conduct of software verification and validation

Outline:

1. Purpose
2. Referenced Documents
3. Definitions
4. Verification and Validation Overview
   - 4.1 Organization
   - 4.2 Master schedule
   - 4.3 Resources summary
   - 4.4 Responsibilities
   - 4.5 Tools, techniques, and methodologies
5. Life Cycle Verification and Validation
   - 5.1 Management of V&V
   - 5.2 Concept Phase V&V
   - 5.3 Requirements Phase V&V
   - 5.4 Design Phase V&V
   - 5.5 Implementation Phase V&V
   - 5.6 Test Phase V&V
   - 5.7 Installation and Checkout Phase V&V
   - 5.8 Operation & Maintenance Phase V&V

Software Verification and Validation Plan

Outline (cont.):

6. Software Verification and Validation Reporting
7. Verification and Validation Administrative Procedures
   - 7.1 Anomaly reporting and resolution
   - 7.2 Task iteration policy
   - 7.3 Deviation policy
   - 7.4 Control procedures
   - 7.5 Standards, practices, and conventions

Source : IEEE

LECTURE NUMBER: 022

TOPIC(S) FOR LECTURE:
Testing

INSTRUCTIONAL OBJECTIVE(S):

1. To understand the role of testing within verification and validation and the software life cycle.
2. To recognize the different types of testing and the purpose of each.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)
(Learning Label- Today we are going to learn ...)

In previous classes, we have talked about the production of quality software and the use of verification and validation. Today we will be examining one aspect of verification and validation in detail -- testing.

CONTENTS:

L22OH1
1. Testing is often *confused* with verification and validation; however, testing is only one component of the verification and validation.

   Testing, according to the IEEE definition, is "the process of exercising or evaluating a system by manual or automatic means to verify that it satisfies specified requirements or to identify differences between expected and actual results." Testing is designed to reveal defects. It shows where a system is correct and where it is wrong. Note that testing and debugging are different; testing reveals the existence of defects while debugging locates and corrects them. In the 1960's and 1970's, it is estimated that over 50 percent of development time and development costs were spent on testing. During this time, people were attempting to test quality into the software instead of building quality into software from the beginning of the life cycle as verification and validation attempts to do now.

L22OH2
2. Discuss the principles of testing which were presented by Mynatt. L22OH3 Note the difference of emphasis from the IEEE definition, Mynatt focuses on error detection while the IEEE definition focuses on showing correctness. Point out that the first four steps can and

should be done as a part of requirements analysis. If a clear set of tests cannot be written during analysis, the indicates that the requirements are not clear.

L22OH4

3. The three primary types of testing are unit or module testing, integration testing, and acceptance testing. Unit or module testing is performed during the implementation phase by the programmer who is building the module.

   a. The primary goals of unit/module testing are to ensure that the module operates correctly and that it carries out its intended function, and to identify the presence of defects. This is generally done by the implementor.

   b. Integration testing is the testing of groups of integrated modules (or subsystems) or the entire system. The most common types of integration testing are design unit and system testing. Design unit testing first uses design information in selecting the modules to integrate and test. This is based on the structure chart. System testing is testing of the whole system. This is an internal acceptance test in a simulated environment. The goals of integration testing are to determine if the subsystem of modules or system meets requirements and functions properly and to test the interface among the modules or subsystems.

   c. Acceptance testing is performed on the finished product in the operational environment. This type of testing is carried out by the sponsor/customer. The goal of acceptance testing is to demonstrate that the system is ready to use and that it meets all the customer's requirements and satisfies all acceptance criteria.

   L22OH5

   c. Discuss the interaction between the various types of testing.

L22OH6

4. A primary concern in performing testing is the generation of test cases. You want to generate enough test cases to thoroughly exercise the software but not so many test cases as to make thorough analysis of the test results impossible. Exhaustive testing, which tests every input and exercises every line of the software, is the technique to use to thoroughly test software, but it is computationally impossible and time-wise too expensive. Alternatives to exhaustive testing include functional analysis, structural analysis, and data-structure-based analysis. These are three complementary approaches to the

generation of test cases.

L22OH7

5. Functional testing, black box testing,is based on functionality, inputs, and outputs of a module with no regard to the internal workings of the module. Three techniques for deriving test cases for functional testing are equivalence partitioning, cause-effect strategy, and boundary values strategy.

   Discuss examples for each technique for deriving functional test cases and work through the exercises with the class. L22OH8, L22OH9, L22OH10, L22OH11, L22OH12,L22OH13, L22OH14, L22OH15, L22OH16. SUggested partitions for L22OH11 include: first character alphabetic and non-alphabetic; length less than 6, greater than 10, and between 6 and 10 inclusive; valid and invalid characters; and passwords which are and are not in the dictionary.

6. Structural testing is based on the internal logic of a module. L22OH17 The concept of coverage analysis is used in structural testing. The types of coverage include statement, decision, condition, and path. Discuss examples L22OH18, L22OH19, L22OH20, L22OH21, L22OH22

7. In performing integration testing, common methods of integration are used in grouping the modules or subsystems for testing. These methods are top-down testing, bottom-up testing, thread testing, and stress testing. L22OH23

8. Many testing and debugging tools are available in today's development environments. Some of these tools include static analysis tools, dynamic analysis tools, test data generators and oracles, file comparators, and simulators. L22OH23

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

Lecture and overheads are the chief media for this lecture.

<u>vocabulary introduced</u>:

testing
unit testing/module testing
integration testing
design unit testing

debugging
system testing
acceptance testing
exhaustive testing
functional analysis
structural analysis
data-structure-based testing
equivalence partitioning
cause-effect strategy
boundary values strategy
statement coverage
decision coverage
condition coverage
path coverage
top-down testing
bottom-up testing
thread testing
stress testing
static analysis tools
dynamic analysis tools
test data generators
test oracles
file comparators
simulators

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>

L22OH1	Testing
L22OH2	Steps of Testing
L22OH3	Principles of Testing
L22OH4	Types of Testing
L22OH5	V&V Testing Activities
L22OH6	Generation of Test Cases
L22OH7	Functional Testing
L22OH8	Equivalence Partitioning
L22OH9	Examples of Equivalence Partitioning
L22OH10	Test Matrix for Equivalence Partitioning
L22OH11	Exercise on Equivalence Partitioning
L22OH12	Cause-effect Strategy
L22OH13	Example of Cause-effect Strategy
L22OH14	Test Matrix for Cause-effect Strategy
L22OH15	Boundary Value Analysis
L22OH16	Examples of Boundary Value Analysis
L22OH17	Structural Testing
L22OH18	Statement Coverage
L22OH19	Example of Statement Coverage
L22OH20	Decision Coverage
L22OH21	Condition Coverage

L22OH22 Testing Strategies
L22OH23 Testing and Debugging Tools

<u>handouts</u>:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 019 - Preliminary test plan presentation/review

READING ASSIGNMENTS:

Sommerville Chapter 23 (pp. 441-454)
Sommerville Chapter 24 (pp. 457-473)
Mynatt Chapter 7 (pp. 274-316)

RELATED READINGS:

Booch Chapter 23 (pp. 420-421)
Booch(2) Chapter 19 (p. 402)
Ghezzi Chapter 6 (pp. 260-293)
Pressman Chapter 18 (pp. 595-626)
Schach Chapter 12 (pp. 385-416)

Testing

The process of exercising or evaluating a system by manual or automatic means to verify that it satisfies specified requirements or to identify differences between expected and actual results

An aspect of verification and validation

Steps of Testing

1. Select what is to be measured by the test

2. Decide how whatever is being tested is to be tested

3. Develop the test cases

4. Determine what the expected or correct result of the test should be and create the test oracle

5. Execute the test cases

6. Compare the results of the test to the test oracle

Principles of Testing

Testing is the process of executing a program with the intention of finding errors.

It is impossible to completely test any non-trivial module or any system.

Testing takes creativity and hard work.

Testing can prevent errors from occurring.

Testing is best done by several independent testers.

Types of Testing

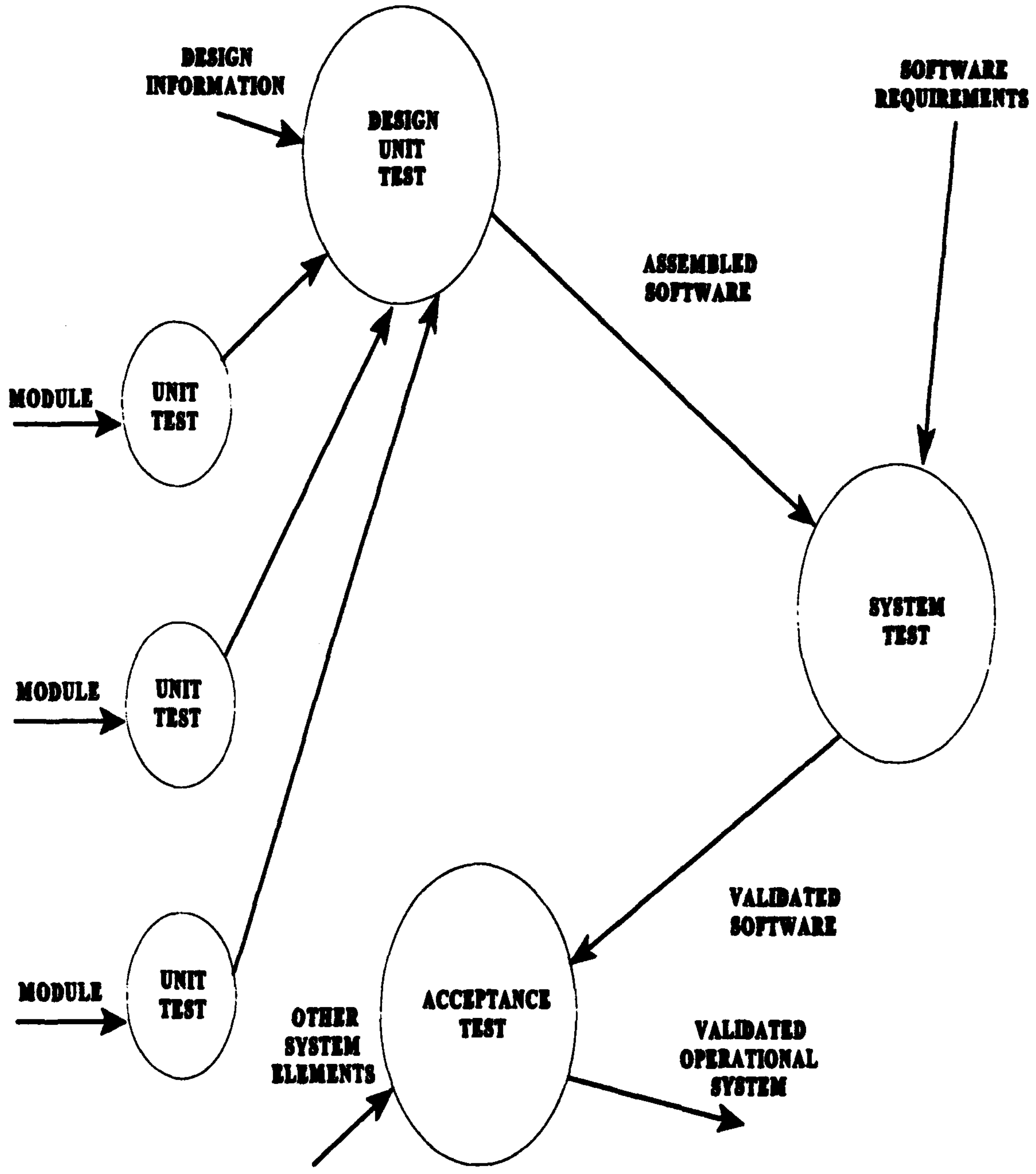
Unit testing/module testing

Integration testing

- Design unit testing
- System testing

Acceptance testing

V & V Testing Activities



Generation of Test Cases

Exhaustive testing

Alternatives to exhaustive testing:
- Functional analysis
- Structural analysis
- Data-structure-based testing

Functional Testing

The specification of external behavior is used to derive test cases

Also called black-box method

Techniques to deriving test cases:
- Equivalence partitioning
- Cause-effect strategy
- Boundary values strategy

Equivalence Partitioning

A equivalence partition consists of a class or set of data items all of which are similar to each other on some relevant dimension

Divide input/output into finite number of equivalence partitions

> Take each input/output condition and divide it into 2 or more groups -- valid equivalence partitions and invalid equivalence partitions

Test one item from each partition

Examples of Equivalence Partitioning

Specifications for Customer Number

Customer numbers must be within the range 1-32700 inclusive without 9 in the unit positions

Partition into:

non-numeric value and numeric value

test groups

1-32700

< 1

> 32700

9 in unit position

< 9 in unit position

Test Matrix for Equivalence Partitioning

Using the specification of the Customer Number:

Partitions or equivalence classes

Number value	1	non-numeric
	2	numeric
Range	3	1-32700
	4	< 1
	5	> 32700
Unit Position	6	9
	7	< 9

Equivalence	Test Cases						
Class Entries		1	2	3	4	5	....
1				X			
2	X	X	X				
3			X				
4		X					
5	X						
6					x		
7			X				

Test Cases:

1. 32708 4. 1AB

2. 0 5. 009
3. 2708

Exercise of Equivalence Partitioning

Validate_New_Password accepts a password from a user and validates that it conforms to the following rules:

1. A password must be between 6-10 characters inclusive
2. The first character must be alphabetic
3. The remaining characters may be any character except control characters
4. The password must not be in a dictionary

Exercise: Develop the test matrix for equivalence partitioning.

Cause-effect Strategy

Tests combinations of inputs

Causes (inputs) and effects (outputs) are identified

Example of Cause-effect Strategy

Determine_Max_Load accepts the GPA for a student and the level of the student (upperclass or lowerclass) and calculates and outputs the maximum class load which the student may take during one semester. If the student has a GPA of 4.0, he/she has a maximum class load of 20. If the student has a GPA of 3.5 or better, he/she has a maximum class load of 18. If the student has a GPA of less than 3.5 and is an upperclassmen, he/she has a maximum class load of 18. If the student has a GPA of less than 3.5 and is an lowerclassmen, he/she has a maximum class load of 16.

Causes

GPA	4.0
	3.5 or higher
	< 3.5
Level	upperclass
	lowerclass

Effects

20
18
16

Test Matrix for Cause-effect Strategy

	Test Cases					
Causes	1	2	3	4	5	6
GPA 4.0	X	X				
GPA >= 3.5			X		X	
GPA < 3.5				X		X
upperclass	X		X			X
lowerclass		X		X	X	
Effects						
20	X	X				
18			X		X	X
16				X		

Test Cases:

1. Upperclass with 4.0
2. Lowerclass with 4.0
3. Upperclass with 3.75
4. Lowerclass with 3.4
5. Lowerclass >= 3.5
6. Upperclass >= 3.5

Boundary Value Analysis

Boundary conditions are the situations directly on, above, and below the edges of input equivalence classes

Include:

- < Beginning element
- Beginning element
- Middle element
- End element
- > end element

Examples of Boundary Value Analysis

Specifications for Calculating Pay

Compute gross pay including overtime rate for hours over:

Partitions	< 40
	= 40
	> 40

Boundary values are 0 and 40

Test:

< 0
0-40
> 40

Sample of test values:

-1	outside of low boundary,
0	at the low boundary,
20	middle value,
40	at upper boundary,
41	above upper boundary

Structural Testing

Approach to testing where the internal logic of a module is used to derive test cases

Also called white-box or glass-box testing

Techniques to derive test cases:

- Statement coverage
- Decision coverage
- Condition coverage
- Path coverage

Statement Coverage

Develop test cases such that every statement is executed at least once

May be too much test data

Does not test all logic (e.g., combination of statements

Example of Decision Coverage

```
1.  if (ORDER >= 20) or (CUSTOMER = 'G')
        then  DISCOUNT := 10
    else if (ORDER >= 10) or (CUSTOMER = 'E')
        then  DISCOUNT := 7
    else if (ORDER < 20) and (CUSTOMER = 'V')
        then DISCOUNT := 5
    else
            DISCOUNT := 0;


2.  if CUSTOMER = 'G'
        then VIP := true
    else
          VIP := false;
```

Test Cases:

1. 20 and 'G' => DISCOUNT = 10 and VIP = true
2. 12 and 'E' => DISCOUNT = 7 and VIP = false
3. 12 and 'V' => DISCOUNT = 5 and VIP = false
4. 9 and 'S' => DISCOUNT = 0 and VIP = false

Decision Coverage

Develop test cases such that each branch is traversed at least once

What are examples of branch statements?

Does decision coverage satisfy statement coverage?

Condition Coverage

Develop test cases such that all combinations of truth values in a decision takes are tested at least once

What are examples of conditions?

Does condition coverage satisfy decision coverage?

What must be added to decision coverage to change it to condition coverage?

Testing Strategies

Top-down testing

Bottom-up testing

Thread testing

Stress testing

Testing and Debugging Tools

Static analysis tools

Dynamic analysis tools

Test data generators and oracles

File comparators

Simulators

LECTURE NUMBER: 023

TOPIC(S) FOR LECTURE:

More on the Structured Analysis model

INSTRUCTIONAL OBJECTIVE(S):

1. Understand the concept and representation of control flows and control transforms in DFDs.
2. Understand use of process specifications and control specifications to describe primitive transforms.
3. Know that a wide range of methods for writing process and control specifications exist and be familiar with a number of them, including narrative English, pseudocode, decision tables, decision trees, graphs, functions, finite state machines, and Petri nets.

SET UP, WARM-UP:

(How involve learner: recall, review, relate)

You are familiar with many aspects of the structured analysis model, having used it in your first project. You produced a CD, a leveled and balanced set of DFDs, and an integrated data dictionary.

(Learning Label- Today we are going to learn ...)

Today we're going to discuss some additional aspects of that structured analysis model.

CONTENTS:

1. Review the structured analysis model already presented and discuss some further aspects.

   a. Context diagram shows the net inputs and net outputs of the system. It shows no decomposition of the system. It is the first level of the model and depicts the relationship between the system and the sources and destinations of the system's inputs and outputs.

   b. DFDs

      i Naming of transforms - the transforms represent actions (functions) and should be named as meaningful as possible yet in only a few words. The name consists of an action verb indicating the function to be performed followed by an object (noun) or adjective/noun. Generic names (e.g. Process input, or Handle transaction) that convey little knowledge of the transform's function must be avoided.

ii Naming of data flows - data flows are named vectors representing "data in motion". They must also be named concisely but meaningfully. Data flow names are always nouns.

iii Control flows - a data flow that represents an element of control (a flag, switch, command signal, etc.) is called a control flow. Generally control flows are not shown (for example, every transform has a "trigger" that invokes it) but at times, particularly in real-time systems, control flows are modeled. A control flow is indicated as a dashed vector. A transform that handles only control flows may be represented by a dashed circle rather than a solid circle. Such a transform is called a control transform.

iv Extensions for real-time systems - There are a number of extensions to the basic structured analysis notation intended to better model real-time systems. Discuss these briefly in order to let students know that these extensions exist and that, without them, real-time systems cannot be adequately modeled. Pressman provides an excellent discussion of these extensions.

v Placement of data stores - are they shown on multiple levels? where are they shown? In general, a data store is shown on the diagram in which it first appears as an interface between two transforms.

vi Review concept of decomposition (leveling), balancing.

2. Discuss how one knows when to stop leveling.

a. Leveling continues until a transform is identified that cannot be further decomposed. A transform that cannot be further decomposed is called a functional primitive, or simply a primitive. While there are no rules for recognizing a primitive, there are a number of guidelines. Discuss these:

i the transform is a simple, obvious, or well-known function and further decomposition is clearly unnecessary

ii the transform has a single input and a single output

iii the policy governing the transform can be easily and clearly described on a single page

b. A process specification is then written for each primitive. Process specifications are also referred to as process specs, p-specs, or mini-specs. A process specification provides a detailed explanation of the internal processing policy of a primitive. Discuss some forms of process specifications that students are already familiar with, e.g. pseudocode.

3. Process specifications (P-specs or mini-specs) - There are many forms of representing process specifications with varying levels of formality. Use examples to illustrate a variety of these notations.

   a. Narrative English is perhaps the most common form but is rarely, if ever, the best form. Illustrate using the verify credit transform described in narrative form in L23OH1.

   b. L23OH2
   Discuss the pseudocode version of Verify Credit and how it is a significant improvement. Pseudocode, or structured English, allows logic to be stated clearly and unambiguously. While informal, standards should be imposed on pseudocode including use of the basic control structures, and reference to data dictionary entries.

   c. L23OH3, L23OH4, L23OH5
   Discuss further examples; different styles.

   d. L23OH6, L23OH7
   Discuss the narrative and data flow diagram description of (L24OH6) a transform to decide on approval for a loan. This is more formally and more effectively represented in a decision table, as shown in L21OH7.

   e. L23OH8
   Discuss the same example represented as a decision tree.

   L23OH9 Another decision tree example.

   f. Other forms - Discuss following as examples of other forms for process specifications and stress that for a given transform, the most appropriate form should be chosen.
      i L23OH10 Finite state machine example
      ii L23OH11a, L23OH11b, L23OH12 Petri net examples
      iii L23OH13 Warnier diagram example (for data store specification)

PROCEDURE:

teaching method and media:

vocabulary introduced:

Control flows
Control transforms
Petri net
Decision trees
Decision table
Psuedocode
Functional primitive
Finite state machine
Warnier Orr diagram

INSTRUCTIONAL MATERIALS:

overheads:

L23OH1	Narrative - Verify credit transform
L23OH2	Psuedocode Example - verify credit
L23OH3	Another Psuedocode Example -
L23OH4	Data Dictionary Example
L23OH5	Structured Requirements Specification
L23OH6	Transform from DFD with narrative
L23OH7	Decision table for loan qualification
L23OH8	Decision tree
L23OH9	Decision tree showing flow of control
L23OH10	Finite State Machines
L23OH11a	Petri nets
L23OH11b	Petri nets
L23OH12	Execution of a Petri net
L23OH13	Warnier diagram

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 020 - Final requirements presentation/review

READING ASSIGNMENTS:

Sommerville Chapter 4 (pp. 71-82)
Mynatt Chapter 2 (pp. 62-69)

RELATED READINGS:

Ghezzi Chapter 5 (pp. 160-198)

Pressman Chapter 7 (pp. 207-235)
Schach Chapter 7 (pp. 157-193)

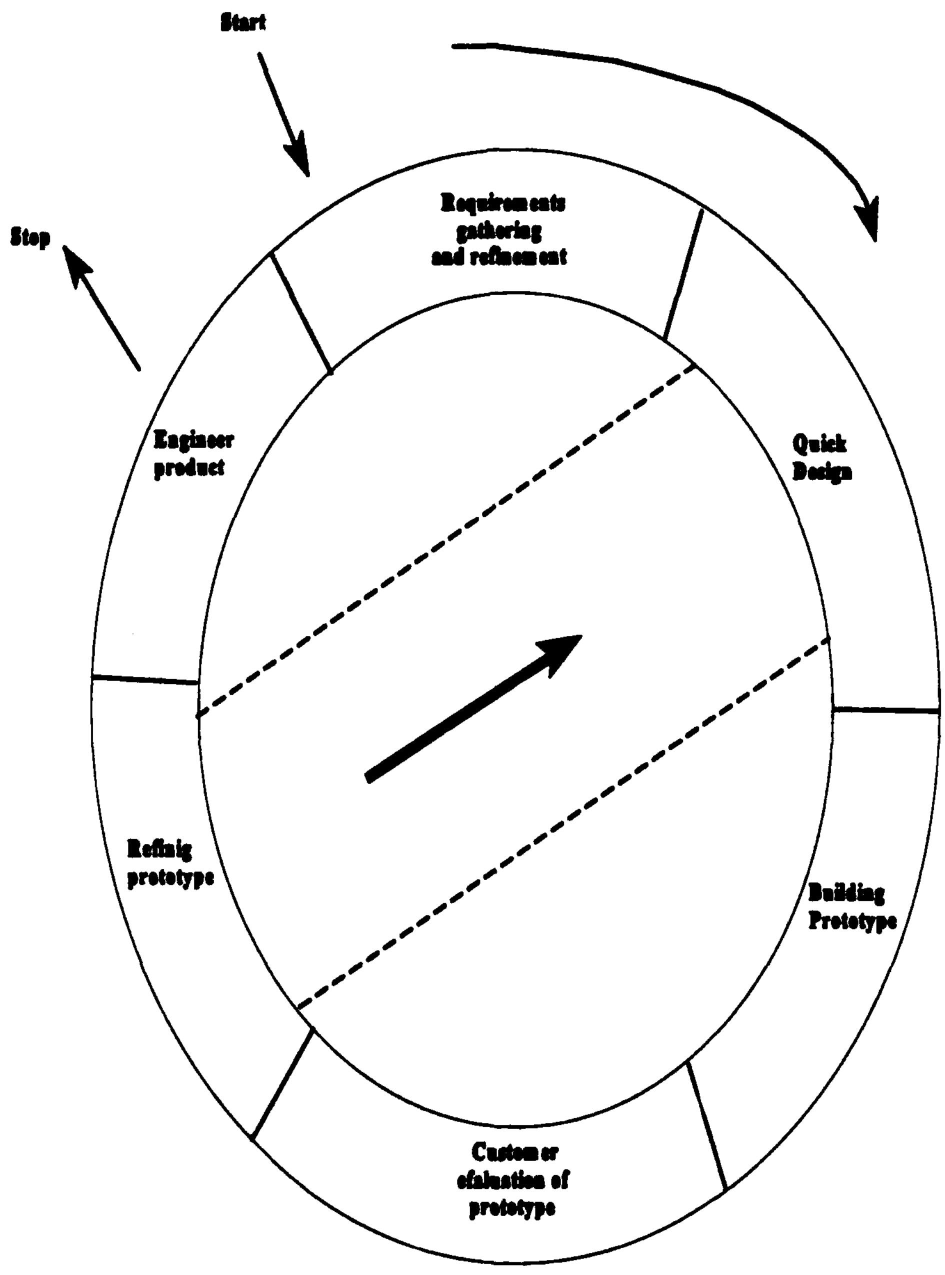
Narrative Description of <u>Verify Credit</u>

"I gather together all of the orders that accumulated the previous day. First I look them all up in customer payment history file and I pull out whatever histories I can find. Then I go through all the histories and add up the overdue balances and mark off the date of the oldest balance. Then I separate the records into two piles. The ones that have no overdue balance or even a balance up to $100 go to Sally. She also gets the ones that aren't more than 60 days old regardless of the overdue balance. She ok's credit for them. All the rest go to Jim who demands prepayments."

Structured English Version of <u>Verify Credit</u>

```
FOR each Order

    Look up Customer_Payment_History for
    Customer_Name (on Order)

    IF no record found (new customer)
    THEN Issue a Prepayment_Request

    ELSE (existing customer)
        Compute Overdue_Balance

        IF Overdue_Balance > 100
        THEN    IF Age_Of_Oldest_Balance > 60
                days

                THEN Issue
                Prepayment_Request

                ELSE (overdue less than 61 days)
                        Issue
                        Credit_Confirmation

        ELSE (overdue balance < $101)
            Issue Credit_Confirmation
```

Prototyping (Pressman)



Plan to Throw One Away

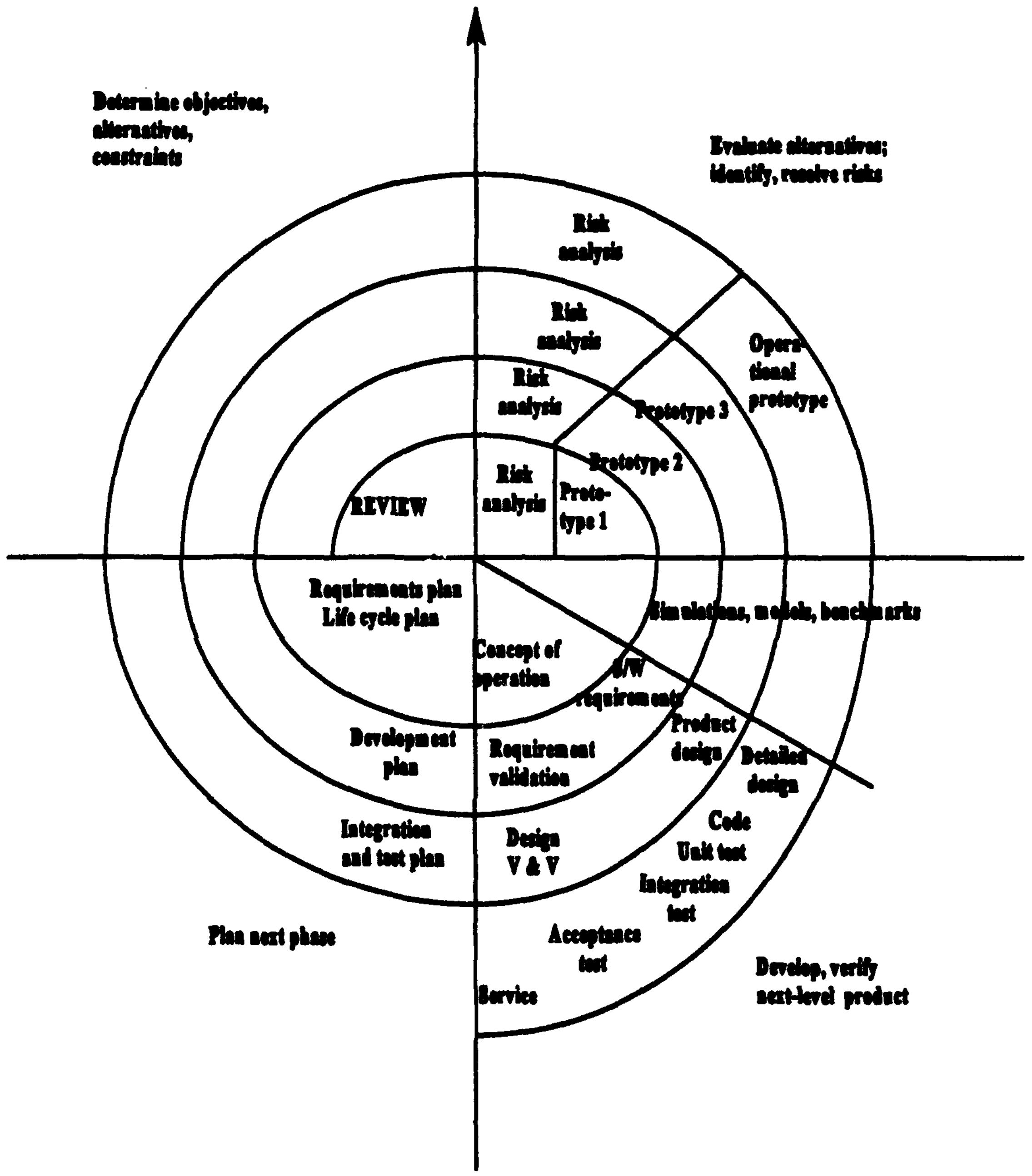
In most projects, the first system built is barely usable. It may be too slow, too big, awkward to use, or all three. There is no alternative but to start again, smarting but smarter, and build a redesigned version in which these problems are solved. The discard and redesign may be done in one lump, or it may be done piece-by-piece. But all large-system experience shows that it be done. Where a new system concept or technology is used, one has to build a system to throw away, for even the best planning is not so omniscient as to get it right the first time.

The management question, therefore, is not *whether* to build a pilot system and throw it away. You *will* do that. The only question is whether to plan in advance to build a throwaway, or to promise to deliver the throwaway to customers. Seen this way, the answer is much clearer. Delivering that throwaway to customers buys time, but it does so only at the cost of agony for the user, distraction for the builders while they do the redesign, and a bad reputation for the product that the best redesign will find hard to live down.

Hence *plan to throw one away; you will, anyhow.*

Source: Brooks: The Mythical Man-Month, 1975, p. 116.

The Spiral Model (Sommerville)



LECTURE NUMBER: 015

TOPIC(S) FOR LECTURE:

Requirements analysis & specification structure
An introductory discussion of requirements identification

INSTRUCTIONAL OBJECTIVE(S):

1. Understand the goals of requirements and their position in the life cycle.
2. Understand the stages in requirements development.
3. Be able to use language syntax, context questions and elicitation to develop requirements.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

In working on your projects you have no doubt come to realize the importance of a clear understanding what is to be developed before starting to build it. In software development, the first step toward such a clear understanding takes place in the requirements process. The completion of this process is marked by the development of a requirements document, sometimes called a software requirements specification.

(Learning Label- Today we are going to learn ...)

Today we are going to learn about the stages of the requirements process and the structure of a requirement document.

CONTENTS:

1. Understand requirements in terms of its 3 primary goals. The first goal is paramount during early stages of the life cycle but diminishes later when the other two goals become more important.

   a. To establish agreement about a system between the sponsors, users and developers of a system.

   b. To function as a transition from the problem space into the solution space by being the basis for software design.

   c. To support the verification and validation of the system.

2. The requirements process and document address the goals of requirements listed above.

a. Requirements definition is the process of determining requirements for a system. The audience here is generally the user and the contractor.

   i Software requirements are distinguished from system requirements when the software requirements are part of a larger system.

   ii L15OH1
   As a formal description of the understanding between customer and software developer, introduce the documentation required by the Department of Defense (DOD) software standard 2167a. This also shows the standards for formally validating these documents.

b. The details of the initial requirements are elaborated in a requirements specification. Sometimes this is divided into high level requirements specification which talks about systems details and software specification which is a document addressed to system designers. Sometimes these are stated formally, allowing a high degree of testability.

c. Requirements validation - refer back to L15OH1.

3. There are several distinct tasks which must be accomplished to identify the requirements. This process is referred to as requirements definition.

   a. Requirements are generally elicited from people. The more complex a system is, the more likely it is multiple people are involved and, thus, multiple viewpoints exist.

   L15OH2 illustrates the use of syntax as a guide to identify the actual requirements. Discuss each of these items using Koff as an example.

   b. Linguistic analysis is only a starting point. Other techniques are required as well. Ask the class for techniques they would use to develop the requirements for a medical diagnosis system. Bring out the need for interviewing experts, understanding the environment (including the equipment), the technicians using the equipment, and the user of the output of the system. Then discuss the techniques below.

      i Context analysis as a method of identification asks why the software is created, what is the environment of the software, and what are the operational, economic boundary conditions that acceptable software must satisfy. The result of this is called <u>software needs</u> or system needs and results in a needs report which should be included in a software requirements specification.

ii Elicitation of requirements related information from end-users, subject matter experts and customers. This is both a fact finding and validation effort. Fact finding includes interviews, questionnaires, and observation of the environment. Validation involves presenting results, including documentation and prototypes, to the customer and resolving open issues.

iii Using system requirements such as those associated with embedded software.

iv Developing user interface requirements. Prototyping may be useful here. It is important to talk to the user rather than the customer or sponsor at this point.

c. Identification of software development constraints: cost, hardware,fault tolerance.

d. There are several other tasks in requirements analysis. They involves relating all of the requirements gathered from diverse sources. One needs to:

i assess potential problems;

ii classify the requirements into categories, determine a method to represent the requirements, and select validation techniques. These will be discussed in later lectures.

PROCEDURE:

teaching method and media:

Lecture and overheads

vocabulary introduced:

functional requirements
non-functional requirements
context analysis
requirements elicitation
linguistic analysis

INSTRUCTIONAL MATERIALS:

overheads:

L15OH1	Software requirements analysis - DoD standard 2167A
L15OH2	Requirements identification through linguistic analysis

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 013 - Peer/self assessments and acceptance reviews for small projects.

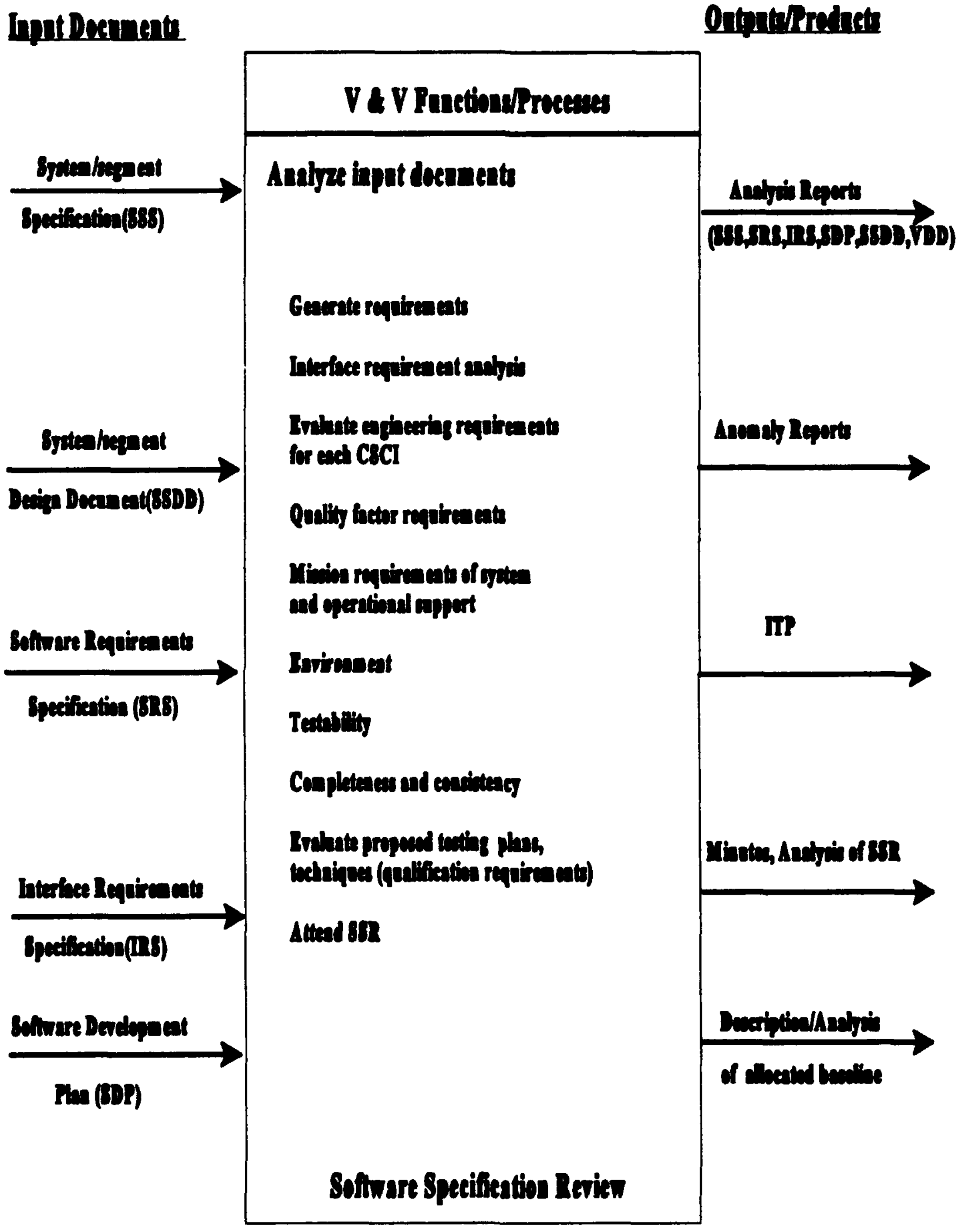
READING ASSIGNMENTS:

Sommerville Chapter 5 (pp. 85-103)
Mynatt Chapter 2 (pp. 62-83)

RELATED READINGS:

Berzins Chapter 2 (pp. 23-47)
Pressman Chapter 6 (pp. 173-189)

Software Requirements Analysis



Input Documents
Outputs/Products
V & V Functions/Processes
System/segment Specification(SSS)
Analyze input documents
Analysis Reports (SSS,SRS,IRS,SDP,SSDD,VDD)
Generate requirements
Interface requirement analysis
Evaluate engineering requirements for each CSCI
System/segment Design Document(SSDD)
Anomaly Reports
Quality factor requirements
Mission requirements of system and operational support
Software Requirements Specification (SRS)
ITP
Environment
Testability
Completeness and consistency
Evaluate proposed testing plans, techniques (qualification requirements)
Minutes, Analysis of SSR
Interface Requirements Specification(IRS)
Attend SSR
Software Development Plan (SDP)
Description/Analysis of allocated baseline
Software Specification Review

Requirements Identification Through Linguistic Analysis

Statements with the verbs "shall", "will", "must", "are", or "is".

Statements that specify numbers, for example, limits, ranges of values, or tolerances.

Statements that are pre-defined or declarations of requirements.

Statements that specify constraints.

Statements that specify size.

Statements that define dependencies, relationships, sequence, logical flow, or behavior.

Statements that specify interfaces, inputs, outputs, events, or interrupts.

Statements that define the data.

Statements that define support.

Statements that specify the environment.

Statements that specify human processing.

Statements that imply requirements, for example, prerequisites.

LECTURE NUMBER: 016

TOPIC(S) FOR LECTURE:

Ada as a specification tool and a maintenance tool

INSTRUCTIONAL OBJECTIVE(S):

1. Understand the structure of Ada systems from program reading of an example.
2. Understand the interface between Ada packages.

SET UP, WARM-UP:

(How involve learner: recall, review, relate)

In a previous lecture, we discussed some of the language features of Ada which support design and maintenance activities. We described how to work from the products of the analysis phase (i.e., requirements list, CD, DFDs, data dictionary) to derive subsystems which compose the proposed system. These subsystems contain information and/or tasks to be accomplished in the system and can usually be classified as user utilities, resource management utilities, and service utilities. Using this approach to high-level architecture of the system allows a high degree of data and functional abstraction. These subsystems can then be shown in Ada package specifications. The design of the interface for these packages or the subprograms within the packages allows you to establish what information flows to and from these subsystems and how the subsystems interact.

(Learning Label- Today we are going to learn ...)

Today we are going to look at another problem specification, its analysis, and its design.

CONTENTS:

1. L16HD1

   Hand out and discuss the narrative description of the spell checker problem.

   Using the context diagram (L16OH1) and the first level DFD (L16OH2), discuss how to determine the subsystems. Possible subsystems (and their actions) include:

   a. Main dictionary (look up a word, keep a counter)

   b. Fast dictionary (look up a word, build, keep a counter)

   c. Auxiliary dictionary (look up a word, add a word, keep a counter)

d. Word (get from line, look up in dictionaries, handle unknown word, keep a counter)

e. Line (get from file, keep a counter)

2. L16HD2
Provide a possible solution -- distribute the Ada package specifications, the driver, and one example of a package body.

Using L16OH3, show the students how these packages and procedures interact. Discuss the design of the software system as shown in these packages. Look at the cohesiveness and coupling provided.

3. Examine the Ada packages for the spell checker, including program features of Ada and the maintainability aspects of the program. Generate discussion through questions about making modifications; e.g. what packages would be affected if changes were made to the package specification of Counter, Test_ops, and Main_dict or the package body of Test_ops.

PROCEDURE:
teaching method and media:

vocabulary introduced:
package specification and body

INSTRUCTIONAL MATERIALS:
overheads:

L16OH1	Context diagram for Spelling checker
L16OH2	Level 0 DFD for Spelling checker
L16OH3	Structure chart for Spelling checker

handouts:

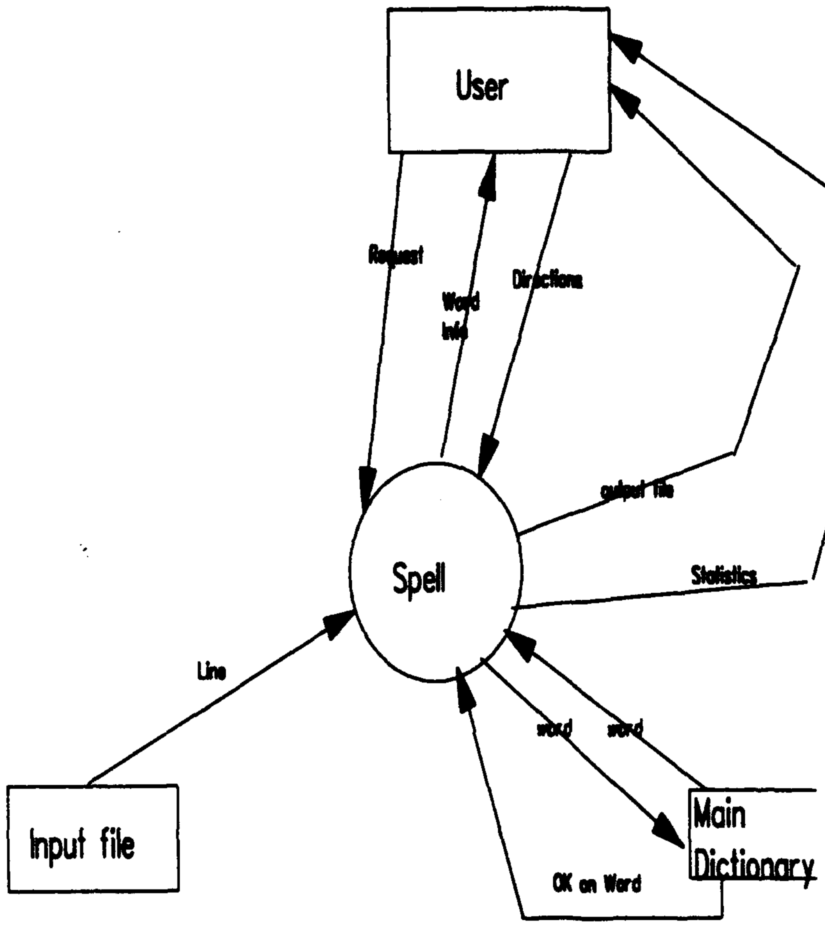
L16HD1	Spelling checker requirements
L16HD2	Ada package specifications, driver, and example of package body

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab014	small project assessment

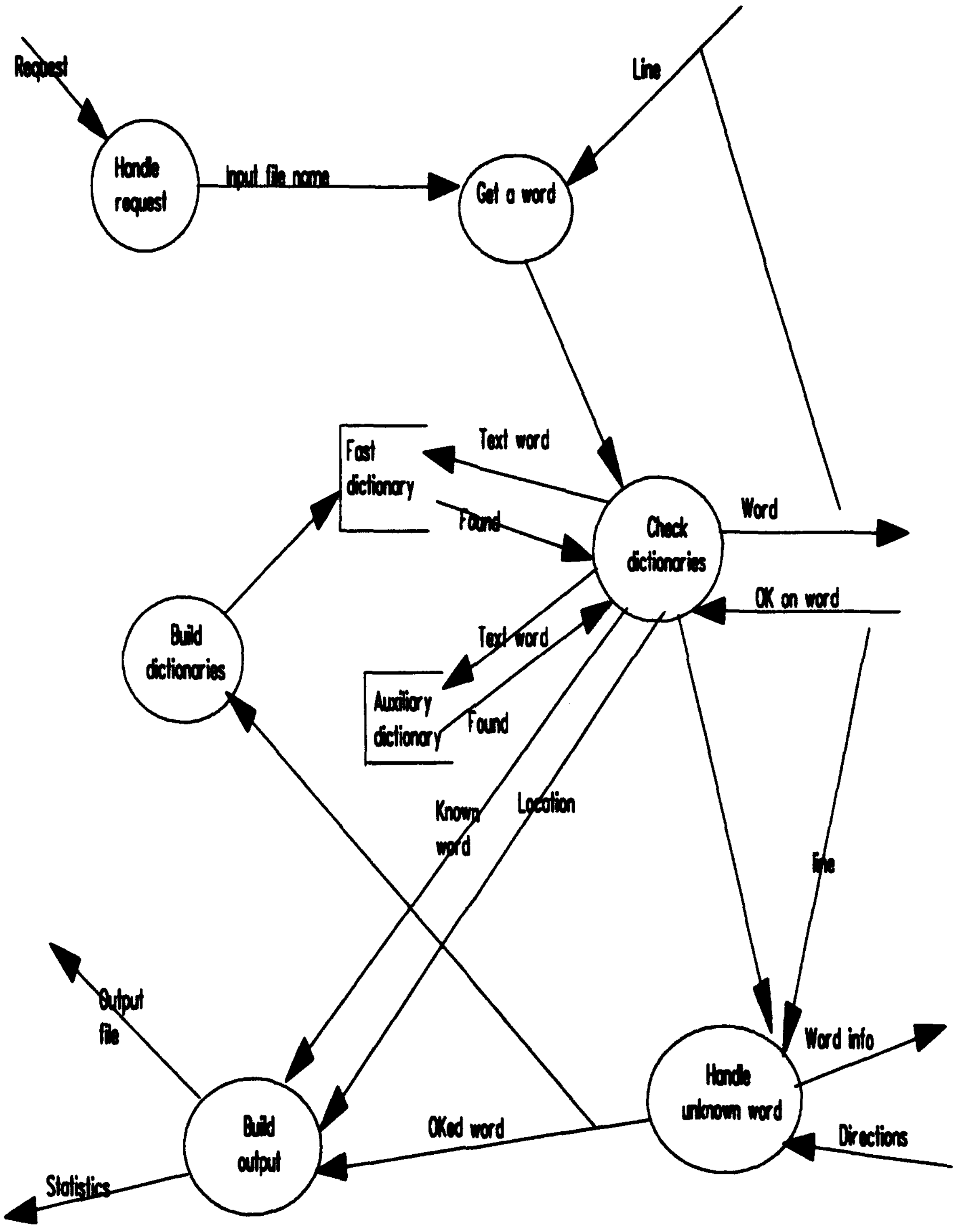
READING ASSIGNMENTS
None

Context Diagram
Spell Checker

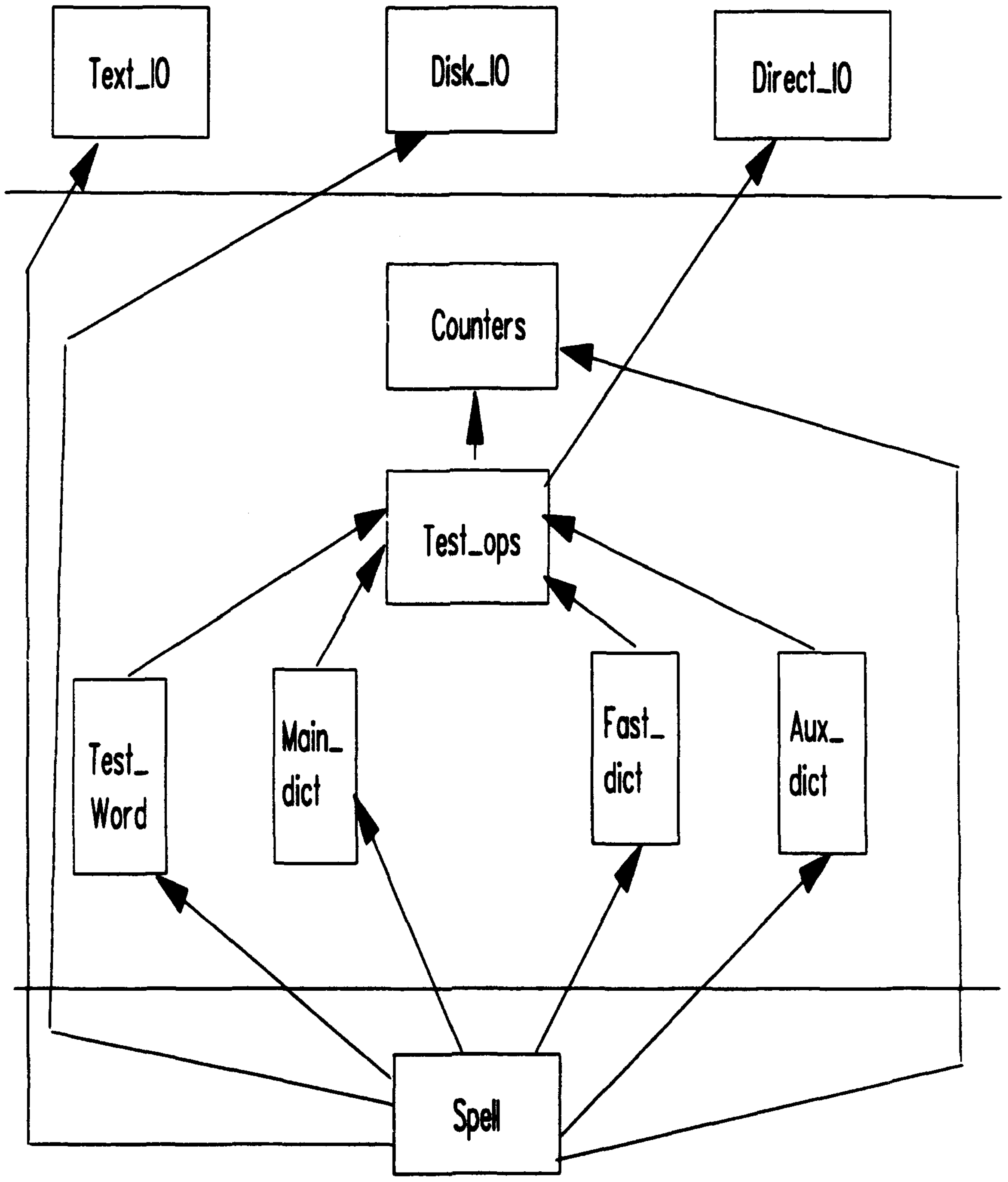


requ est =	command + name of input file
word info =	word + line
direction =	ingnore | add to dictionary | correctly spelled word
statistics =	number of lines processed + number of words processed + number of words found in main dictionary + number of words found in fast dictionary + number of words found in auxiliary dictionary

Level 0
Spell Checker



Structure Chart
Spell Checker



Spelling Checker Requirements

Narrative:

Spell is a general-purpose spelling checker that operates on an existing editor-created text file to produce an output text file that has been checked for spelling errors. Spell parses out the words from an input file and compares them with entries in its dictionaries. Whenever a word is not found in its dictionaries, the checker will indicate the word and the line of text that contains the word and seek the user's directions regarding the word.

Spell features a large permanent dictionary accessed from disk, as well as a small "fast" dictionary that is built and loaded into fast memory and contains the most commonly used words in the English language. In addition, an auxiliary dictionary that contains words inserted by the user is on line. This auxiliary dictionary is the only one that may be modified and maintained by the user. Unknown words that are correctly spelled may be automatically added to the auxiliary dictionary.

In addition to the output file, Spell will provide statistical information on the file which was processed. This information includes the number of lines of text processed, the number of words processed, the number of words found in the main dictionary, the number of words found in the fast dictionary, and the number of words found in the auxiliary dictionary.

Spell is required to run on a microcomputer in an interactive manner. The microcomputer must have, at a minimum, a video terminal, two disk drives (floppy or hard), and 128,000 bytes of random access memory. Spell should be able to process at least 200 words per minute.

```
package COUNTERS is

        type COUNTER is limited private;

        procedure INITIALIZE (C : out COUNTER);
        -- used to initialize an object of type COUNTER to zero

        procedure INCREMENT (C : in out COUNTER);
        -- used to increment an object of type COUNTER by one

        procedure DISPLAY (C : in COUNTER);
        -- used to display the value of an object of type COUNTER

private
        type COUNTER is new integer;
end Counters;
```

```
with Direct_io, COUNTERS;  use COUNTERS;

package TEXT_OPS is
        -- from Direct_io import type File_type
        -- from COUNTERS import type COUNTER

        type LINE is limited private;
        type WORD is limited private;

        INPUT                    : Direct_io.File_type;
        END_OF_LINE, LONG_WORD  :  boolean;
        -- END_OF_LINE is set by GET_LINE to true at the end of a line
        -- LONG_WORD is set by GET_NEXT_WORD to true if word is over
        --   13 characters

        function GET_LINE return LINE;
        -- obtains the next line of text for processing

        function GET_NEXT_WORD (L : LINE) return WORD;
        -- obtains the next word in line for processing

        procedure GET_INPUT_TEXT (NAME : in String);
        -- fetches an existing text file with Name

        procedure CREATE_OUTPUT_TEXT (NAME : in out String);
        -- opens a new text file Name

        procedure PUT_OUTPUT (L : in LINE);
        -- sends line to the output text file

        procedure WORD_HANDLER (W : in WORD; L : in out LINE;
                                          W_COUNT, L_COUNT : in COUNTER);
        -- handles an unidentified word

        function UPPERCASE (W : WORD) return WORD;
        -- converts a word to uppercase

        function SPECIAL_ENDING_1 (W : WORD) return boolean;
        -- returns true if word ends in 'S' or 'D'

        function SPECIAL_ENDING_2 (W : WORD) return boolean;
        -- returns true if word ends in 'ED', 'ER', or 'LY'

        function SPECIAL_ENDING_3 (W : WORD) return boolean;
        -- returns true if word ends in 'EDS', 'ERS', or 'ING'

        function STRIP_ENDING_1 (W : WORD) return WORD;
        -- removes 'S' or 'D' ending from word
        function STRIP_ENDING_2 (W : WORD) return WORD;
        -- removes 'ED', 'ER' or 'LY' ending from word

        function STRIP_ENDING_3 (W : WORD) return WORD;
        -- removes 'EDS', 'ERS' or 'ING' ending from word

        function ING_ENDING (W : WORD) return WORD;
        -- returns true if word ends in 'ING'

        function ADD_E (W : WORD) return WORD;
        -- adds 'E' to word

        procedure DISPLAY_LINE (L : in LINE);
        -- displays a line on a video terminal

        procedure DISPLAY_WORD (W : in WORD);
        -- display a word on a video terminal

        procedure REMOVE_APOSTROPHES (W : in out WORD);
        -- removes first and last apostrophes, if present in
        -- either the first or last character of the word

        function LENGTH (W : WORD) return integer;
        -- returns the number of nonblank characters in the word

private
        LINE_LENGTH : constant := 80;
        WORD_LENGTH : constant := 13;

        type LINE is array (1..LINE_LENGTH) of character;
        type WORD is array (1..WORD_LENGTH) of character;

end TEXT_OPS;
```

```
with TEXT_OPS; use TEXT_OPS;

package TEST_WORD is
        -- from TEXT_OPS import WORD

        function IDENTIFY_WORD (W : WORD) return boolean;
        -- returns true whenever word is found in one of the
        -- available dictionaries

end TEST_WORD;
```

```
with TEXT_OPS; use TEXT_OPS;

package  MAIN_DICT is
        -- from TEXT_OPS import WORD

        NUM_FOUND_MD : integer;
        -- NUM_FOUND_MD is incremented by one every time a word
        --   is found in the main dictionary

        procedure CLOSE_MD;
        -- closes the main dictionary file

        function LOOKUP_MD (W : WORD) return boolean;
        -- returns true if word is found in main dictionary

end MAIN_DICT;
```

```
with TEXT_OPS; use TEXT_OPS;

package FAST_DICT is
        -- from TEXT_OPS import WORD

        NUM_FOUND_FD : integer;
        -- NUM_FOUND_FD is incremented by one every time a word
        --   is found in the fast dictionary

        procedure CLOSE_FD;
        -- closes the fast dictionary file

        procedure BUILD_FD;
        -- builds fast dictionary file from main dictionary

        function LOOKUP_FD (W : WORD) return boolean;
        -- returns true if word is found in fast dictionary

end FAST_DICT;
```

```
with TEXT_OPS; use TEXT_OPS;

package AUX_DICT is
        -- from TEXT_OPS import WORD

        NUM_FOUND_AD : integer;
        -- NUM_FOUND_AD is incremented by one every time a word
        --   is found in the auxiliary dictionary

        procedure CLOSE_AD;
        -- closes the auxiliary dictionary file

        function LOOKUP_AD (W : WORD) return boolean;
        -- returns true if word is found in auxiliary dictionary

        procedure INSERT_AD (W : in WORD);
        -- used to insert a word into auxiliary dictionary

        procedure INSERT_AD_UNUSED (W : in Word);
        -- used to insert a word into the unused portion of
        -- the auxiliary dictionary

end AUX_DICT;
```

```
with TEXT_OPS, MAIN_DICT, FAST_DICT, AUX_DICT;
use TEXT_OPS, MAIN_DICT, FAST_DICT, AUX_DICT;

package body TEST_WORD is
        -- from TEXT_OPS import WORD, UPPERCASE, LENGTH,
        --                          SPECIAL_ENDING_1, SPECIAL_ENDING_2,
        --                          SPECIAL_ENDING_3, STRIP_ENDING_1,
        --                          STRIP_ENDING_2, STRIP_ENDING_3,
        --                                          ING_ENDING,     ADD_E,
REMOVE_APOSTROPHES
        -- from MAIN_DICT import LOOKUP_MD
        -- from FAST_DICT import LOOKUP_FD
        -- from AUX_DICT import LOOKUP_AD, INSERT_AD_UNUSED

        function IDENTIFY_WORD (W : WORD) return boolean is
                TEMP_WORD : WORD;
                I         : integer;

                function IS_IN_DICTIONARIES (W : WORD) return boolean is
                begin
                        if (Length(W) <= 6) and then LOOKUP_FD (W) then
                          return True;
                        elsif LOOKUP_AD (W) then
                          return True;
                        elsif LOOKUP_MD (W) then
                          INSERT_AD_UNUSED (W);
                          return True;
                        else
                          return false;
                        end if;
                end IS_IN_DICTIONARIES;

        begin --IDENTIFY_WORD
                W := UPPERCASE (W);
                -- remove apostrophe if it is first or last symbol
                REMOVE_APOSTROPHES (W);
                if SPECIAL_ENDING_1 (W) then
                  TEMP_WORD := STRIP_ENDING_1 (W);
                  if IS_IN_DICTIONARIES (TEMP_WORD) then
                        return true;
                  end if;
                elsif SPECIAL_ENDING_2 (W) then
                  TEMP_WORD := STRIP_ENDING_2 (W);
                  if IS_IN_DICTIONARIES (TEMP_WORD) then
                        return true;
                  end if;
                elsif SPECIAL_ENDING_3 (W) then
                   TEMP_WORD := STRIP_ENDING_3 (W);
                   if IS_IN_DICTIONARIES (TEMP_WORD) then
                         return true;
                   end if;
                   if ING_ENDING (W) then
                      TEMP_WORD := ADD_E (W);
                      if IS_IN_DICTIONARIES (TEMP_WORD) then
                            return true;
                      end if;
                   end if;
                end if;
                if IS_IN_DICTIONARIES (W) then
                   return True;
                else
                   return False;
                end if;
          end IDENTIFY_WORD;

    end TEST_WORD;
```

```
with  TEXT_OPS,  COUNTERS,  TEST_WORD,  MAIN_DICT,  FAST_DICT,
AUX_DICT,
   TEXT_IO, DISK_IO;
use   TEXT_OPS,  COUNTERS,  TEST_WORD,  MAIN_DICT,  FAST_DICT,
AUX_DICT,
   TEXT_IO, DISK_IO;

procedure SPELL is
     -- from COUNTERS import INCREMENT and DISPLAY
     -- from TEXT_OPS import LINE, WORD, GET_LINE, GET_NEXT_WORD,
     --                    END_OF_LINE, LONG_WORD, INPUT,
     --                    GET_INPUT_TEXT, CREATE_OUTPUT_TEXT,
     --                    PUT_OUTPUT, AND WORD_HANDLER
     -- from TEST_WORD import IDENTIFY_WORD
     -- from MAIN_DICT import NUM_FOUND_MD and CLOSE_MD
     -- from FAST_DICT import NUM_FOUND_FD and CLOSE_FD
     -- from AUX_DICT import NUM_FOUND_AD and CLOSE_AD
     -- from Text_io import put, get, new_line, and put_line
     -- from Disk_io import end_of_file

     package INT_IO is new Integer_IO (integer);
     use INT_IO;

     INPUT_LINE    : LINE;
     A_WORD        : WORD;
     NAME          : String (1..20);
     LINE_COUNT,
     WORD_COUNT    : COUNTER;

     procedure INFORMATION;
     begin
          put ("What is the name of the text to be checked?");
          new_line;
          put ("Your must use the '.TEXT' suffix > ");
          get (Name);
          GET_INPUT_TEXT (NAME);
          new_line (3);
          put ("What is the name of the new text?");
          new_line;
          put ("You must use the '.TEXT' suffix > ");
          get (NAME);
          CREATE_OUTPUT_TEXT (NAME);
     end INFORMATION;

begin
     INITIALIZE (LINE_COUNT);
     INITIALIZE (WORD_COUNT);
     INFORMATION;
```

```
loop
        INPUT_LINE := GET_LINE;
        exit when END_OF_LINE (INPUT);
        INCREMENT (LINE_COUNT);
```

```
          loop
               A_WORD := GET_NEXT_WORD (INPUT_LINE);
               exit when END_OF_LINE;
               if not LONG_WORD then
                 INCREMENT (WORD_COUNT);
                 if not IDENTIFY_WORD (A_WORD) then
                     put (ASCII.BEL);
                       WORD_HANDLER    (A_WORD,    INPUT_LINE,
WORD_COUNT,
                                   LINE_COUNT);
                 end if;
               end if;
          end loop;
          PUT_OUTPUT (INPUT_LINE);
     end loop;
     new_line (3);
     put ("The number of lines processed is ");
     DISPLAY (LINE_COUNT);
     new_line;
     put ("The number of words processed is ");
     DISPLAY (WORD_COUNT);
     new_line;
     put ("The number of words found in main dictionary: ");
     put_line (NUM_FOUND_MD);
     new_line;
     put ("The number of words found in fast dictionary: ");
     put_line (NUM_FOUND_FD);
     new_line;
     put ("The number of words found in auxiliary dictionary: ");
     put_line (NUM_FOUND_AD);
     new_line;
end SPELL;
```

LECTURE NUMBER: 017

TOPIC(S) FOR LECTURE:

1. **Requirements as an end product and the standards applied to them.**
2. **The requirements development process.**
3. **Requirements validation.**

INSTRUCTIONAL OBJECTIVE(S):

1. Use several techniques to extract requirements.
2. Understand several methods of requirements specification.
3. Follow the steps in a requirements standard such as 2167a.
4. Develop a requirements validation plan.

SET UP, WARM-UP:

(How involve learner: recall, review, relate)

As you recall we have discussed several techniques for identifying requirements. Once the requirements are identified there are still several major tasks which have to be accomplished.

(Learning Label- Today we are going to learn ...)

Today we are going to learn about the elements that make up the remaining stages of the requirements process and how they relate to a requirements document. We will discuss the elements of a requirements document and the steps in the process of developing the document.

CONTENTS:

1. Ask the students to recall the 3 primary goals of a requirements document.

   a. To establish agreement about a system between the sponsors, users and developers of a system.

   b. To function as a transition from the problem space into the solution space by being the basis for software design.

   c. To support the verification and validation of the system.

2. The development of requirements is quite difficult and several organizations have formulated complete software development methodologies which include a discussion of requirements.

   a. Requirements definition: as the process of determining requirements for a system. The software development process has been formalized by a number of organizations, Private Companies, NASA, DoD. We will look at one of those standards DoD_STD 2167A.

      i Software requirements are distinguished from system requirements when the software requirements are part of a larger system.

L17OH1

ii Discuss the document structure endorsed by 2167a. The relation is systems contain segments, segments contain configuration items, configuration items contain configuration components(CSCs) and components contain units(CSUs).

iii This standard treats software development as a milestone-driven project. Milestones are generally documents or clearly specified events. 2167a characterizes several processes separated by the completion of milestones. L17OH2
Point out the reviews. Work through the chart showing the relation to what steps they have experienced in the small project.

b. Briefly discuss functional requirements and tasks to be accomplished. Functional requirements are the external behavior(the functions) expected by the user of the system. These tasks must be clearly stated in a precise manner so that they can be tested. For example a functional requirement of the KoFF system would be dispense tapes.

c. Discuss non-functional requirements in terms of restrictions or constraints on the system. These are elements which the customer lives with all the time and so they are the hardest to extract from him/her. They are caused by hardware, laws, the environment, oolitical preferences. Sometimes these requirements are very hard to gather. Ask the students why there might be more difficulty gathering these requirements. (The sort of answer you are looking for includes "The customer takes their environment for granted and presumes that you understand the constraint of their system.) For example - the customer forgot to tell you that their hardware is a Commodore 64 computer with 64 K RAM.

i Metrics for non-functional requirements include speed, size, ease of use, reliability, robustness, portability

ii Review the Mynatt list on pages 71 and 72 & give examples.

d. Ask the students about non-functional requirements of their small projects. [E.g., the system is to be implemented in Pascal, the KoFF system must release tapes within 3 seconds of selection. Note that the function to be performed -dispense tapes- has not changed.] Another example is writing a system for a foreign customer, where the documentation must be in their language.

e. Requirements Specification presents the details of the system. Sometimes this is divided into high-level requirements specification which talks about systems details and software specification which is a document addressed to system designers. There are many ways to express these details. Natural language has several ambiguities. Sometimes the same requirements get listed as two tasks because the developer did not realize it was the same task. Formalism have been developed to try to reduce ambiguity. Examples of such formalism are formal-algebraic specification and special languages such as PSL/PSA and SADT.

f. Specifications should include all the default conditions and how to handle error conditions. Often default conditions are forgotten by the analyst or the customer. In these cases when the system is executed, it is initialized to a default state no one considered, which could be unpredictable and or dangerous. Because the analyst is not completely familiar he/she does not know how error conditions should be handled. All error conditions and how to handle them should be specified in the requirements.

g. Requirements validation - it is critical to get the requirements correct because any mistake made here will cost more in effort and dollars later on.. There are many standards for requirements. They should be checked for consistency, correctness, and completeness, feasibility, functionality, testability, easy to change

   i Formal review - walk-throughs and inspections

   ii Requirements validation matrix-compare this with the requirements traceability matrix used in the sample test plan L9HD1.

3. There are several distinct tasks which must be accomplished to develop a requirements definition.

   a. Remind them of the discussion of requirements identification and ask them to describe the three methods of requirements gathering discussed in the last class.

      i Context analysis as a method of identification (from Ross) which asks why the software is created, what is the environment of the software, and what are the operational, economic boundary conditions that acceptable software must satisfy. The result of this is called software needs.

      ii Elicitation of requirements related information from end-users, subject matter experts and customers. This is both a fact finding and validation effort. Fact finding includes interviews, questionnaires, and observation of the environment. Validation involves presenting documentation of the results to the customer and resolving open issues.

b. There are several other inputs to the process of requirements gathering.
   - i Using system requirements such as those associated with embedded software)
   - ii Developing user interface requirements (2167a Interface Requirements Specification IRS)

c. Identification of software development constraints: cost, hardware,fault tolerance.

d. Requirements analysis involves relating all of the requirements gathered from diverse sources- satisfy the customer
   - i Assessment of potential problems and determination of acceptable risk
   - ii Classification of requirements done in terms of mandatory, desirable, and inessential. It is also helpful to classify them in terms of stability- which ones are likely to change. Ask them how this knowledge would effect design. They should isolate requirements that are likely to change.
   - iii Consider both the technical(can computers do it) and operational(can the staff use the system in our environment) feasibility of the system. Also consider the economic feasibility of the system. What would the student think of developing a checkbook balancing program which required input using reverse polish notation for a computer that weighed 20 pounds and cost $350.00.

e. Requirements representation- a step of requirements definition which portrays the results of requirements identification.
   - i Use of models
   - ii Prototyping as a means of clarifying the requirements

f. Requirements communication involves presenting the requirements to diverse audiences for review and approval

g. Requirements validation
   - i Show 2167a requirements summary evaluation criteria and go over them. L17OH3
   - ii Show 2167a figure 5 and review the items L17OH4

Traceability to system specification and statement of work. Consistency with IRS and other specifications for interfacing items. Testability of requirements. Adequacy of quality factor requirements.

Traceability to system specification and statement of work. Consistency with other specifications for interfacing items. Testability of requirements.

iii Discuss verification and validation(V&V) as a separate process and then show and discuss the V&V standards for requirements analysis. L17OH5

iv Establishment of acceptance criteria

iv Tie this to development organizations which is the subject of the next class. Some methods of organizing software development teams improve the quality of the validation.

PROCEDURE:

teaching method and media:

The class was primarily lecture with some discussion.

vocabulary introduced:

2167a
Requirements validation plan
Computer software configuration item (CSCI)
Computer software components (CSC)
Computer software units (CSU)
Hardware configuration items (HWCI)
Interface requirements specifications (IRS)

INSTRUCTIONAL MATERIALS:

overheads:

L17OH1	The language of Standard 2167A
L17OH2	Deliverable products
L17OH3	Requirements summary evaluation criteria
L17OH4	Software requirements analysis
L17OH5	Verification and validation standards for requirements analysis

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 015 - Initial user perspective of extended project

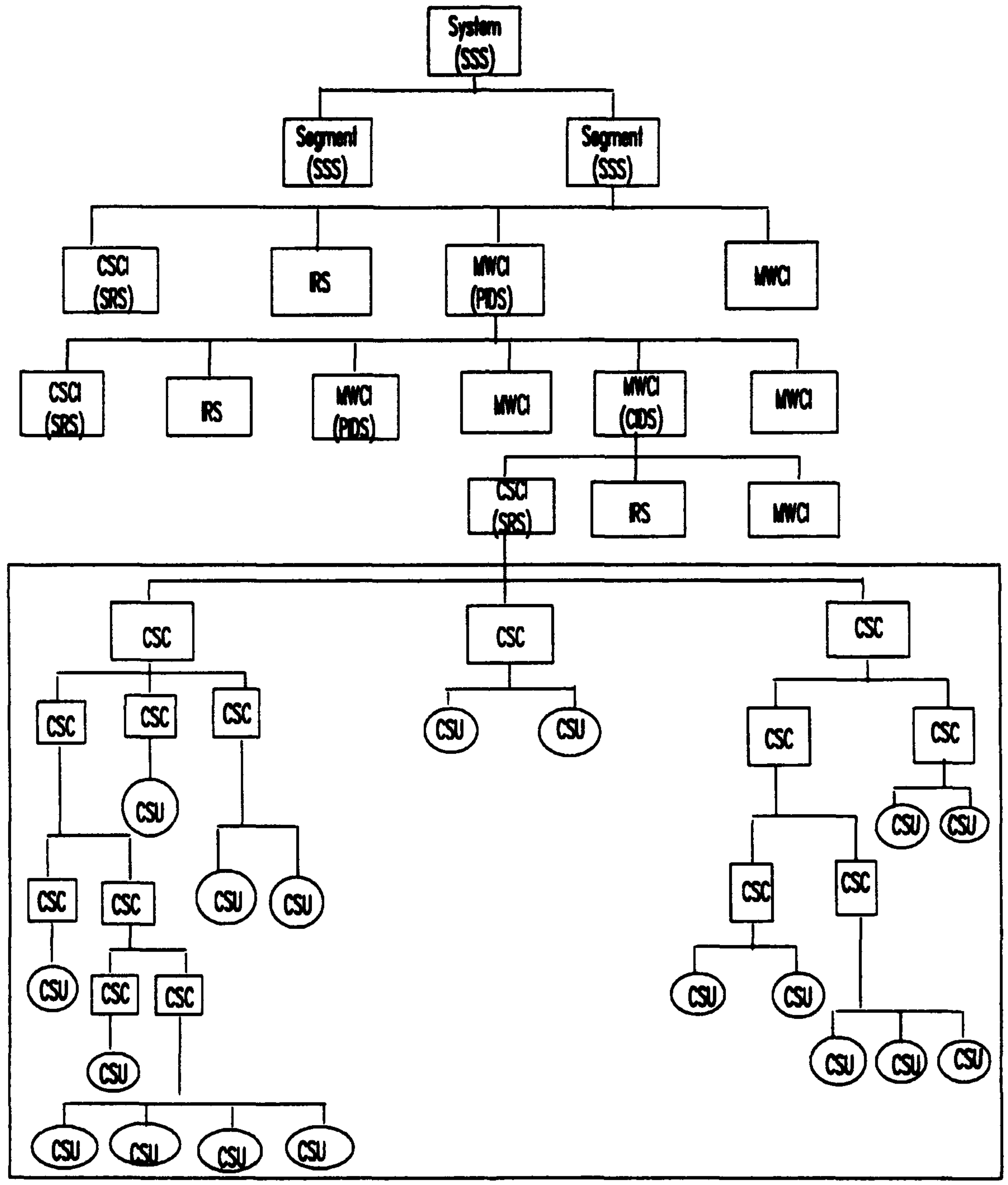
READING ASSIGNMENTS:

Sommerville Chapter 3 (pp. 45-61)
Sommerville Chapter 5 (pp. 85-91)
Mynatt Chapter 2 (pp. 62-91)

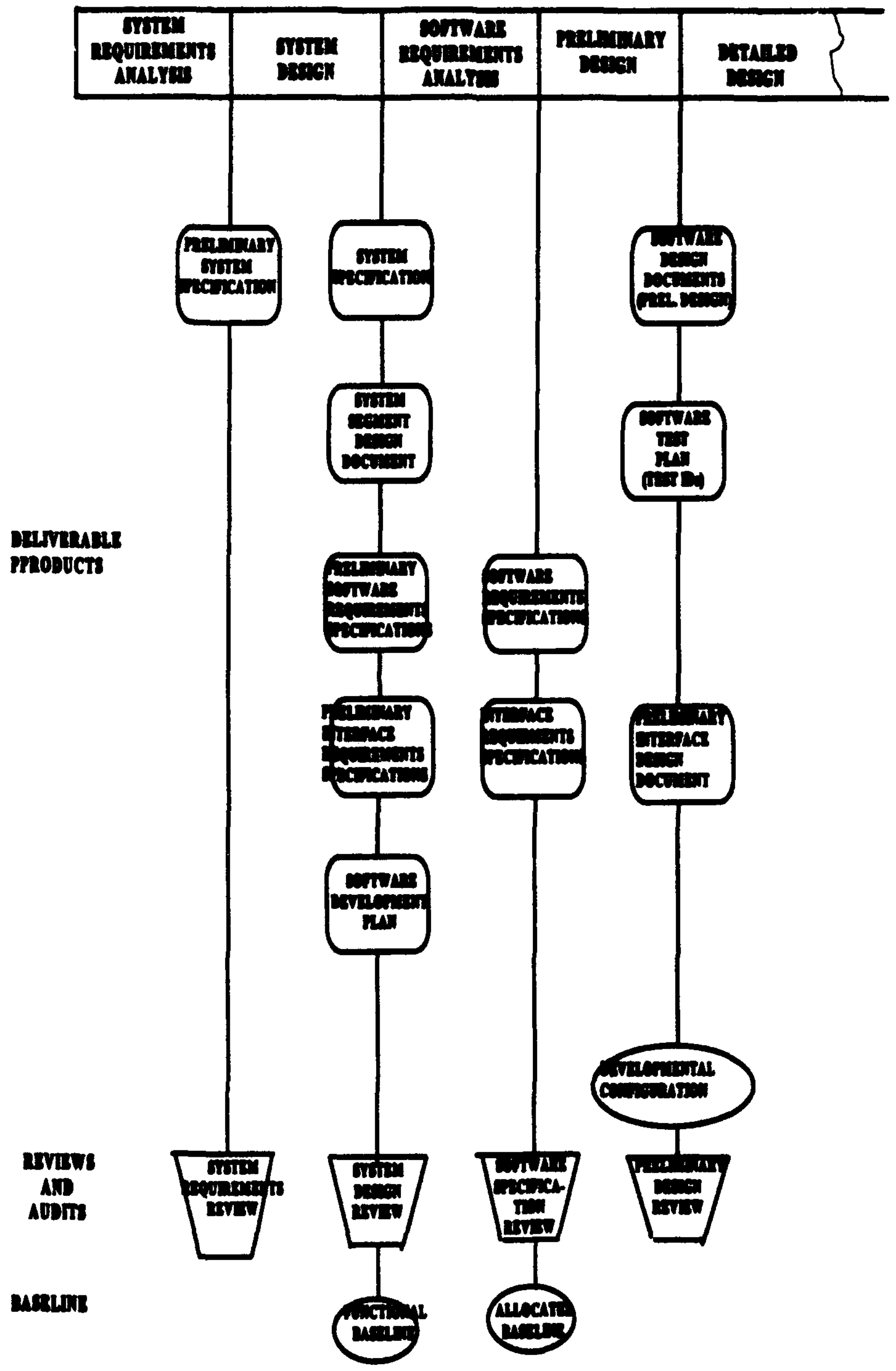
RELATED READINGS:

Ghezzi Chapter 5 (pp. 151-160)
John Brackett, Software Requirements, SEI-CM-19-1.2, January 1990.

DoD-STD-2167A Example of System Breakdown and CSCI Decomposition



DoD-STD-2167A
Deliverable Products, Reviews, Audits and Baselines


SYSTEM REQUIREMENTS ANALYSIS
SYSTEM DESIGN
SOFTWARE REQUIREMENTS ANALYSIS
PRELIMINARY DESIGN
DETAILED DESIGN
DELIVERABLE PPRODUCTS
PRELIMINARY SYSTEM SPECIFICATION
SYSTEM SPECIFICATION
SOFTWARE DESIGN DOCUMENTS (PREL. DESIGN)
SYSTEM SEGMENT DESIGN DOCUMENT
SOFTWARE TEST PLAN (TEST IDs)
PRELIMINARY SOFTWARE REQUIREMENTS SPECIFICATIONS
SOFTWARE REQUIREMENTS SPECIFICATIONS
PRELIMINARY INTERFACE REQUIREMENTS SPECIFICATIONS
INTERFACE REQUIREMENTS SPECIFICATIONS
PRELIMINARY INTERFACE DESIGN DOCUMENT
SOFTWARE DEVELOPMENT PLAN
DEVELOPMENTAL CONFIGURATION
REVIEWS AND AUDITS
SYSTEM REQUIREMENTS REVIEW
SYSTEM DESIGN REVIEW
SOFTWARE SPECIFICATION REVIEW
PRELIMINARY DESIGN REVIEW
BASELINE
FUNCTIONAL BASELINE
ALLOCATED BASELINE

Requirements Evaluation Criteria

Criterion	Attributes
Language	■ Concise, quantitative requirements ■ Proper use of Shall and Will
Consistency	■ Standardized format ■ Technical consistency ■ Uniform level of detail
Completeness	■ Acceptable technical level ■ Timing, accuracy requirements stated ■ Capacities specified ■ Terms and acronyms defined
Lack of ambiguity	■ Clear organization ■ Firmness of requirements ■ Clear functional traces possible ■ Requirements not open to interpretation
Necessity	■ Requirements needed to fulfill system objectives ■ Requirements not superfluous
Testability	■ All the above criteria satisfied ■ Capability to develop physical and functional tests

Evaluation Criteria for Products of Software Requirements Analysis

Internal Consistency

Understandability

Traceability to the indicated documents

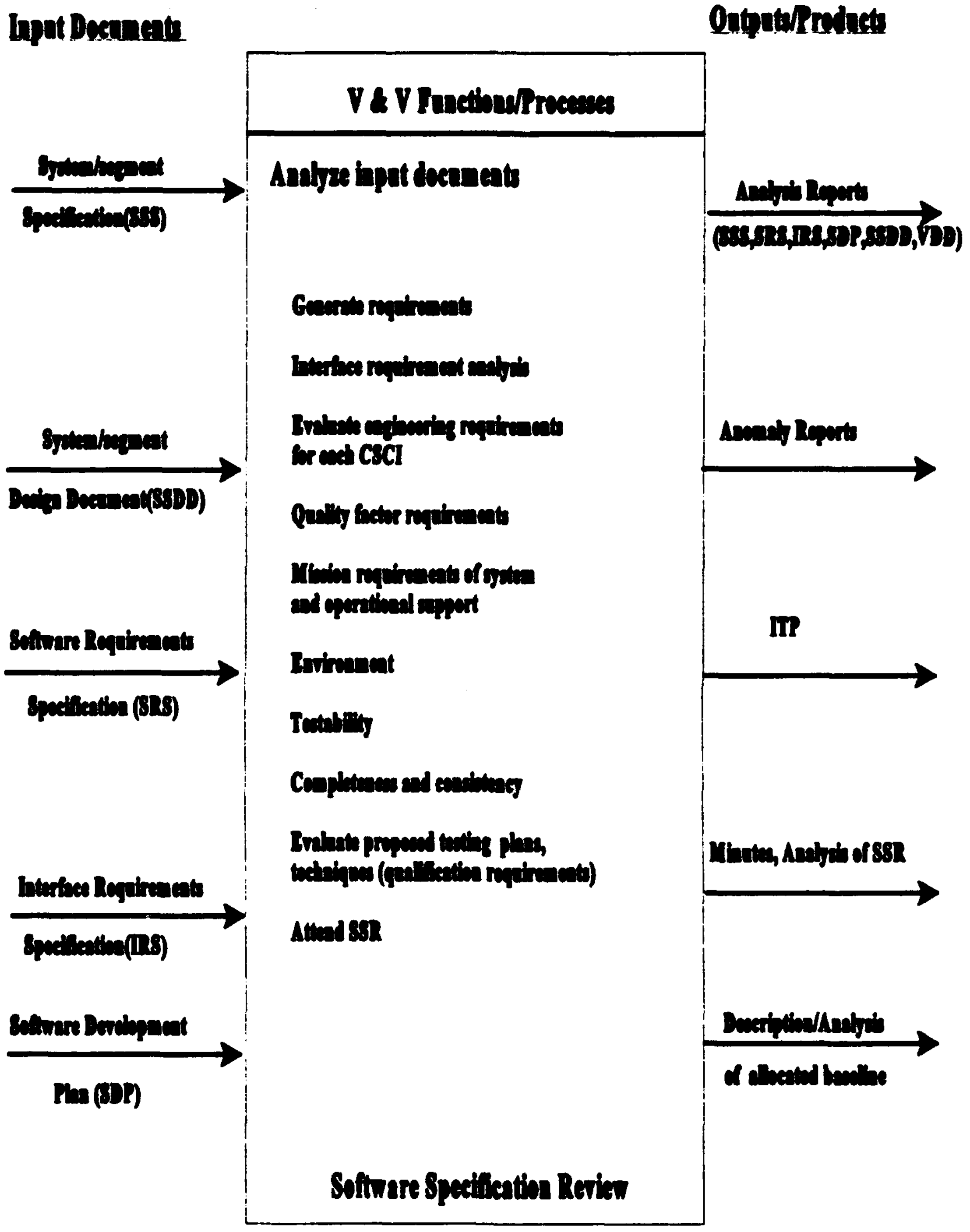
Consistency with the indicated documents

Appropriate analysis, design, or coding techiques used

Appropriate allocation of sizing and timing resources

Adequate test coverage of requirements

Software Requirements Analysis


Input Documents
Outputs/Products
V & V Functions/Processes
System/segment Specification(SSS)
System/segment Design Document(SSDD)
Software Requirements Specification (SRS)
Interface Requirements Specification(IRS)
Software Development Plan (SDP)
Analyze input documents
Generate requirements
Interface requirement analysis
Evaluate engineering requirements for each CSCI
Quality factor requirements
Mission requirements of system and operational support
Environment
Testability
Completeness and consistency
Evaluate proposed testing plans, techniques (qualification requirements)
Attend SSR
Software Specification Review
Analysis Reports (SSS,SRS,IRS,SDP,SSDD,VDD)
Anomaly Reports
ITP
Minutes, Analysis of SSR
Description/Analysis of allocated baseline

LECTURE NUMBER:018

TOPIC(S) FOR LECTURE:

1. **Team organizations and software quality.**
2. **Roles and responsibilities in a matrix organization.**

INSTRUCTIONAL OBJECTIVE(S):

1. **Understand different project team organizations and their impact on quality.**
2. **Understand the variety of roles in software development.**

SET UP, WARM-UP:

(How to involve learner: recall, review, relate)

Most software projects today are too large to be completed by a single individual. Several people have to work together in completing individual tasks which contribute to the final software product, and several groups of people have to work together organizing the components which constitute the final complex software artifact. The structure, effective organization, and management of these teams has a direct impact on the quality and timeliness of the final software product.

(Learning Label- Today we are going to learn ...)

Today we are going to look at different types of organizations and their impact on quality.

CONTENTS:

1. Team organization- Software development like any management effort must organize people so that they can effectively accomplish their goals. The structure of the team is dictated by its goals. Discuss how sports teams are organized. Each member of a team has a specific role and has particular constraints placed on him/her. The third baseman is encouraged to handle the ball when it is hit or thrown to him/her. However the third baseman is prohibited from pitching the ball.

2. Teams can be organized based on control or function.

   a. We characterize team organization based on where decision making control resides. A team can have centralized control where a recognized leader is responsible for all final decisions and to resolve all technical issues. One such model is called a chief programmer team. A team organization can also be based on a distributed model of control, emphasizing group consensus. This is illustrated in a democratic team organization. There can also be hybrid combination of these two types of control.

b. We can also define an organization in terms of its functions. Large organizations have a control structure which ia tied to the major functions of the organization. For example consider an organization whose primary goal is sales. It will have departments for marketing sales and publicity but not for manufacturing. The primary decision making responsibility will be distributed to each of these departments.

3. Four basic organizational structures that have been used to model teams after are:

a. An application organization is a traditional hierarchical organization with clearly visible product objectives. The chain of command and control is vertical. This structure has the advantages of isolating the lower levels from higher level decisions.

b. A functional organization is organized around technical skills of expertise. A system testing group or an analysis group represent a functional unit in a functional organization. Function groups, like application groups, normally work on many projects. This is a service oriented structure rather than a product oriented structure. This has the problem that because a manager directs several projects, which project is only a part time effort. Reporting in this group is not to the project manager directing the project but to the functional group manager.

c. A project organization is a single group formed to carry out a single long term project. Because this group is dedicated to a single project it has significant visibility.

d. A matrix organization is organized on two axes -- one being skill groups and the other being projects. The vertical axes consists of groups such as test, code, requirements, etc. While the horizontal axes consists of the projects currently under review or in progress. This model enables the allocation of human resource to multiple projects. This is useful for temporary or short lived projects. Personal who are already familiar with the environment will get assigned to the project. However this type of organization does not foster devotion to an individual project.

L18OH2

4. Several software team organizations can be used within these

business organization.

a. The democratic team in which there are no predetermined lines of communication is a common model of team. This is essential the model followed on your first project. Ask the students what problems exist for this type of team. You are looking for things like: no clear decision maker and so work had to be redone and significant difficulty in communication. For each additional team member, we reduce the communications capability. The total number of lines of communication in this type of team is N(N-1)/2.

b. The chief programmer model, sometimes called the surgical model established very clearly defined roles and threads of control. The team is made up of about six programming support personnel, one of who is the backup for or assistant to the chief programmer who could replace the chief programmer if necessary. The chief programmer designs and implements the critical parts of the software. The other members of the team consist of an administrator who takes care of the day to day non-programming details; a librarian who maintains all the program listings and can function as project configuration manager. The team will also have a toolsmith or language specialist who cal take can of critical language decisions. In this model communication is minimized through the administrator. If a new programmer is added then there is only one new line of communication to the administrator.

c. There are hybrids of these two teams which minimize communication using a surgical model and then open the lines of communication at the more technical level using the democratic model.

5. Assessment of team organization

a. Different teams are appropriate to different projects, no one team organization is appropriate for all tasks.

i. decentralized control works well when communication at a low level is needed to achieve the goal.

ii. centralized control works well when the problem is well-understood and rapid development is important.

b. Discuss the advantages and disadvantages of each type of team organization.

PROCEDURE:

teaching method and media:

vocabulary introduced:

functional organization
chief-programmer team
centralized control
democratic team
matrix organization

INSTRUCTIONAL MATERIALS:

overheads:

L18OH01
L18OH02

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 016 - Immediate tasks for configuration management, requirements, user interface, and test plan teams.

READING ASSIGNMENTS:

Mynatt Chapter 1 (pp. 31-42)

RELATED READINGS:

Ghezzi Chapter 8 (pp. 440-446)
Schach Chapter 11 (pp. 357-368)

Software Development Teams and Organizations

Team Management Goals:

- Clear assignment of responsibility
- Facilitate cooperation for common goals
    - effective size
    - clear leadership structure

Control based organization

- Centralized control
- Distributed control
- Hybrid forms of control

Functional Based organization

Teams and Risks

Software Orgnaizations:

- Organizational
- Functional

Software Teams

- Democratic Team
- Surgical/Chief Programmer Team
- Hybrid Team

LECTURE NUMBER: 020

TOPIC(S) FOR LECTURE:
Moving from entity relationship diagrams (ERDs) to Ada

INSTRUCTIONAL OBJECTIVE(S):

1. Understand the concepts and notations of ERDs
2. To learn how to use ERDs to derive objects and operations to form an Ada specification.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

In a recent class, you were introduced to ERDs as an analysis notation for understanding the problem domain during structured development. These ERDs can also be used to assist in deriving the objects and operations for a problem domain.

(Learning Label- Today we are going to learn ...)

Today we will look at using ERDs to derive objects and operations.

CONTENTS:

1. Entity relationship diagram (ERD)

   a. An ERD graphically depicts the static nature of the major entities of the system and their respective interrelationships. An ERD can be derived from an event list. An event list is a list of the system's actions which affect the system's processing. This list can be stated in a standard sentence format with a subject, verb, and direct object. An ERD can be derived from an event list by defining the subjects and direct objects as entities and the verbs as relationships between the entities. An entity is some individual item of interest in the problem domain. An ERD is a network which depicts how entities of the system are interrelated.

   b. ERDs are well known and understood and provide a good starting point for the derivation of objects and operations in a problem domain. The derivation of objects and operations in a problem domain is not a well understood process, as of yet.

ERDs provide a static view of a system which can be used as a building block for the data abstraction of object-oriented design. The usefulness of the ERD is that it provides a frame of reference for identifying the objects of the object-oriented design.

2. Entity categories

   a. The entities from the ERD provide a starting place for determining the objects of a system; however, not all entities from the ERD will become objects in the final design. The entities are first categorized before determining which entities become objects in the design.

   b. The following entity categories have been identified as common types of entities which exist for a system:
      - i External entities are the terminators on the context diagram which do not "own" any data in the problem domain. These entities do not require data definition within the scope of the system.
      - ii Internal entities are the terminators on the context diagram which do "own" data in the problem domain. These entities require data definition within the scope of the system.
      - iii User-view entities present a "view" to the user. Examples are a report or a screen.
      - iv Dependent entities have little significance to the system alone and must be associated with another entity for identification.
      - v Identifiable entities are independent system elements.

3. Objects and operations from entities

   a. Any entities which are derived from nonautomated events are eliminated from further consideration.

   b. External entities and dependent entities are eliminated from consideration as objects. External entities are excluded from the design since they are not part of the problem domain. Dependent entities are excluded from the object list since they are represented in the design by other objects upon which they are dependent.

   c. The remaining entities provide a preliminary list of objects for the system.

d. The operations which remain are associated with the operations specified in the ERD. The DFDs for the system may help to identify operations based on events in the ERD.

e. Each object and its associated operations becomes a subsystem (i.e., Ada package specification) for the system. These package specifications will be completed giving the interface for each operation.

PROCEDURE:

teaching method and media:

vocabulary introduced:

INSTRUCTIONAL MATERIALS:

overheads:

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 017 - Configuration management plan presentation/review

READING ASSIGNMENTS:

none

RELATED READINGS:

Stoecklin, Adams, and Smith, "Object-oriented Analysis" at Proceedings of the Fifth Washington Ada Symposium, June 1988, Tysons Corner, Virginia, pp. 133-138.

LECTURE NUMBER: 021

TOPIC(S) FOR LECTURE:

Verification
Validation

INSTRUCTIONAL OBJECTIVE(S):

1. To understand the roles of verification and validation in the software life cycle.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

In previous classes, we have talked about the production of quality software being a major concern in software engineering. A necessary approach to achieve quality software is through the use of verification and validation.

(Learning Label- Today we are going to learn ...)

Today we will be examining verification and validation in detail.

CONTENTS:

1. Introduction to verification and validation (V&V) L21OH1

   a. Prior to the use of verification and validation, testing was the primary means of ensuring the quality of software. However, testing was performed in an informal, arbitrary manner usually by the programmer working in isolation. Testing also occurred only at the end of development after implementation. Quality products are developed during a quality process. People were trying to put quality into the software AFTER the software was developed. They were trying to test quality into the software but this approach was not working.

   b. Verification and validation are process for ensuring quality software throughout the life cycle from requirements through maintenance. V and V are complimentary, yet distinct. The main objectives of V and V are the discovery of defects in any of the products (requirements document, design document, etc) as they are developed; and the assessment of whether or not the system satisfies the specified requirements. This process permeates the entire life cycle. Errors should be detected as early in the development cycle as possible.

L21OH2

c. Three types of analysis are used in V&V. Static analysis involves no execution; it is the manual or automated examination of a product (e.g., software requirements specification document, source code). Examples of static analysis include software reviews and static program analyzers.

   Dynamic analysis involves execution of software where the functional, structural, or computational aspects of the software are examined. Examples of dynamic analysis include unit or module testing and acceptance testing.

   Formal analysis is the use of mathematical techniques to evaluate a product; examples of formal analysis include symbolic execution and proof of correctness.

2. Verification

L21OH3

a. Are we building the product right? Verification involves evaluating the end product of each phase; we are looking for errors generated within the phase and/or by the transformation between phases. The product is evaluated for its consistency, completeness, and correctness according to the previous phase. This is done during each phase and not at the end of the life cycle. The most common errors occur between phases.

b. Although the product is the primary focus, the quality of the development process is also being evaluated.

c. The tasks of verification are to assume that the products of each software life cycle phase:
   i Comply with previous life cycle phase requirements and products,
   ii Satisfy the standards, practices, and conventions of the phase, and
   iii Establish the proper basis for initiating the next life cycle phase activities.

d. Static analysis, dynamic analysis and formal analysis are used in accomplishing verification.

3. Validation L21OH4

	q	r	s	t	u
p	Data	----	Data or stamp	Common	Common
q		Control	Data or stamp	----	----
r			----	Data	----
s				----	Data
t					Common

Qualities of Coupling Levels (Page-Jones)

Coupling Type	Suscepti-bility to ripple effect	Modifia-bility	Under-stand-ability	Module's Usability in other systems
Data	Variable*	Good	Good	Good
Tramp	Poor	Medium	Medium	Poor
Stamp	Variable*	Medium	Medium	Medium
Bundling	Variable*	Medium	Poor	Poor
Control	Medium	Poor[T]	Poor[T]	Poor[T]
Hybrid	Medium**	Bad	Bad	Bad
Common	Bad	Medium	Bad	Poor
Content	Bad	Bad	Bad	Bad

* Depends on the breadth (the number of individual items) of the interface.

T Poor mainly because of concomitant problems in the interface and the cohesion of one of the modules.

** If the convention used in the hybrid data has to be changed, the ripple effect can be devastating.

LECTURE NUMBER: 026

TOPIC(S) FOR LECTURE:

Preliminary design using functional decomposition
Problem solving paradigms
Introduction to object-orientation

INSTRUCTIONAL OBJECTIVE(S):

1. Understand functional preliminary design
2. Define basic concepts of object-orientation
3. Understand an object-oriented approach to analysis

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

L26OH1

We have learned the basic elements of design. The inputs and outputs to the design process have been defined for us. One of our authors -Mynatt- describes the preliminary design process and includes in that process interface design and software design. She and others have described design as "basically a creative process". What does she mean?

(Learning Label- Today we are going to learn ...)

L26OH2

The problem with preliminary design seems to arise at the transitions from analysis to an architecture and from an architecture to a documented solution. Today we will look at some ways to make those transitions easier and concentrate on a new approach called object orientation.

CONTENTS:

L26OH3

1. Discuss where the difficulty lies in design. Define domain analysis as required knowledge of the system environment. Discuss the difficulties they just had in moving from data flow diagrams to structure charts. The translation process between the problem domain and the solution domain is difficult and it is made more difficult by using different notations being used to talk about the problem domain and the solution domain. For example, data flow diagrams for the problem space and structure charts for the solution space. domain.

   In SA/SD attention shifts from analysis to design, the way you look at the problem changes. Although there are techniques to help move from SA to SD, this shift of focus makes the transition more difficult.

L26OH4

2. Discuss how object-oriented design is intended to address this

problem by providing a seamless transition between development stages. The focus of object-oriented design is on the things in the system and these things tend to be more stable throughout the development process. One starts with a set of objects which are easily understood in the analysis stage. Object-oriented design is an elaboration of the way these object are related to form a solution to the initial problem.. The details of the design get embodied into the objects. This encapsulation facilitates maintenance and reusability.

L26OH5

3. Begin an illustration of object oriented development by giving preliminary definitions of Objects, Classes and Inheritance.

4. Spend about 10 minutes with an object identification exercise L26OH6. Brainstorm with the students about the objects that are needed. The objects they identify will include bottle, fill, label, wash, cap, box, ship .... List the objects on the board as they describe them. Pause to show them that they have outlined a system without being language specific. Discuss some of these candidate objects, noting that some of them are both nouns and verbs. Label is an interesting one. A label has data--the type of beverage--, it also has some properties -- glue on one side-- which enable it to undergo the Labeling process. Once they are convinced that they have a high level, language independent, description of a system, develop an Ada package called LABEL L26OH7 to show them how an object can be specified in Ada. Use the package to explain some of the concepts of reuse and encapsulation. If there is time you might want to give a high level treatment of generics and develop a generic package called FILL which is passed two parameters, the size of the bottle to be filled and the type of beverage to be placed in the bottle. Exceptions could be touched on here as the error conditions for this package, e.g., overflow of bottle and not_completely_fill the bottle.

L26OH8

4. Discuss Wenger's distinction between Object-oriented languages and object-based languages. Because Ada does not currently support inheritance it is useful to spend some time talking about the virtues of object-orientation and that many of those virtues can be achieved independent of the implementation language. Work done at the NASA software engineering lab has proven that the use of an object-oriented methodology, independent of the language environment and the availability of inheritance, produces significant benefits. Use L26OH9 to discuss some of the elements of object-orientation. Software engineering starts with real world non-computer objects, e.g., cars, or vending machines. These objects are easily identifiable and are more than just functions or data. Emphasize the black-box nature of these objects. They present and external interface to the work and restrict access to their internal implementations. In Object-

orientation, this is called information hiding makes non-essential information inaccessible. This can be modeled in an Ada package. The body of a package physically encapsulates both data and function.

Talk about data and functional abstraction and how they are combined in object-orientation. Functional abstraction focuses on the interface to the object, but is does not know how the function is accomplished within the object. Using a vendor supplied sort package is a good example of this.

Inheritance can be simply modeled. Tell the students that you have a Rumbo outside. When they ask what it is, tell them it is a car. Point out that now they can tell you several things, both attributes and functions, about a Rumbo because it inherits characteristics from the class CAR. This is a good point to use a sample of Rumbaugh notation. Show a class diagram for car and them under it place two other class diagrams for SEDAN and STATION WAGON. Draw an inheritance relation between these three showing how sedan and station wagon inherit all of the characteristics of car. Use the example of the car and return to the concept of abstraction. The CAR object can be presented at several levels of abstraction. A high level of abstraction views a CAR as an object which transports people. At a lower level we can talk about its structure or the interconnectedness of its part; and at a lower level we can talk of the functions of its parts. These levels of abstraction model the stages of object-oriented development from analysis to preliminary design to detailed design.

L26OH10

5. Review the standard problem solving paradigms and how object-oriented design fits in with these paradigms. The choice of paradigm directs the entire development life cycle. Paying attention to paradigms is a shift in focus from "Let's see how I can solve this problem in 'C' " to "What design technique will best support a solution to this problem?"

6. Begin a discussion of domain analysis and talk about the different categories of objects, viz., physical objects, roles, incidents like airline flights, and interactions between objects like employee works for company. How these object are related in object-oriented design will be the subject of the next lecture.

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

vocabulary introduced:
- objects
- encapsulation
- inheritance
- object-based
- object-oriented
- information hiding
- procedural abstraction
- data abstraction
- functional abstraction

INSTRUCTIONAL MATERIALS:

overheads:

L26OH1	Design
L26OH2	Preliminary design - The Transitions
L26OH3	Preliminary design - The Transitions, Missing Elements
L26OH4	Object-Oriented Design
L26OH5	Object Orientation
L26OH6	An exercise to identify some objects in a problem specification
L26OH7	Ada package specification for Don's Brewery
L26OH8	Object-oriented vs object-based
L26OH9	General characteristics of object orientation
L26OH10	Object-oriented development

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 021 - Preliminary design

READING ASSIGNMENTS:
- Sommerville Chapter 10 (pp. 182-188)
- Mynatt Chapter 3 (pp. 94-130)

RELATED READINGS:
- Berzins Chapter 4 (pp. 207-214)
- Booch Chapter 5 (pp. 44-50)
- Booch(2) Chapter 3 (pp. 34-41)
- Ghezzi Chapter 4 (pp. 115-121)
- Pressman Chapter 8 (pp. 239-262)
- Schach Chapter 9 (pp. 262-264)

Design

Preliminary Design

Input	Software specifications
Process	Generate architecture to meet specifications
Output	Preliminary design document

Detailed Design

Preliminary Design - The Transitions

Input	Problem description in terms of functions

Determine a solution

Process	Preliminary solution in terms of system architecture, might include interface design

Record a solution

Output	Preliminary design document

Preliminary Design - The Transitions Missing Elements

Input Problem description in terms of functions

DOMAIN ANALYSIS: non-functional requirements, understanding of the environment

Determine a solution

How ? - "Basically a creative process", "A Flair"

Process Preliminary solution in terms of system architecture, might include interface design

Record a solution

What method of notation best reflects both the problem and the solution?

Output Preliminary design document

Object-Oriented Design

Goals of Object-Oriented Design

Avoid translation problem between problem and solution statements.

Establish a seamless transition from design to implementation.

Make programming simpler, more like real-life (Alan Kay-Smalltalk)

Facilitate reuse of all system components.

Make all forms of maintenance easier and more reliable.

Object Orientation

Objects	Real world entities (SE) or modules with constructor and inspectors, Ada packages (Programmers)
Classes	A template for similar objects or instances
Inheritance	A class acquires the characteristics from one or more other classes.

An Exercise to Identify Some Objects in a Problem Specification

Don is going to automate one aspect of his brewery. He wants a computerized system to control the bottling of the beverages: lager, ale, stout, and bitters, that he brews. The returnable bottles come in 3 sizes: one pint, two pints and three pints.

Ada Package Specification for Don's Brewery

```
with Text_IO;
uses Text_IO;
with BOTTLE;
uses BOTTLE;

Package LABEL is
     Procedure GET_LABEL;
     Procedure WET_LABEL;
     Procedure PLACE_LABEL;
end LABEL;

Package body LABEL is
..
..
Type
Records
Functions
Procedures
end LABEL;
```

Object-Oriented vs Object-Based

Peter Wenger 1986
The essential elements for object orientation

1. Support for data abstraction
2. Management of data abstraction by typing
3. Composition of abstract data types through an inheritance mechanism

"The benefits of object-orientation have been proven to be dependent on the adoption of object-oriented methodology rather than on the implementation details" Mike Stark - Software Engineering Laboratory

General Characteristics of Object Orientation

Information hiding

Abstraction
- Process
- Entity
- Levels of abstraction
- Functional and data

Encapsulation

Inheritance

Object-Oriented Development

Problem Solving Paradigms

- Procedural
  - Stream of actions
  - Data structures are passed
  - State of system maintained globally
- Logical
- Access-Oriented
- Object-Oriented
  - Problem domain objects
  - Control distributed in objects
  - State maintained by separate objects
- Functional

Change of orientation from coding as a foundation for a solution to the requirements and design as a foundation for a solution

LECTURE NUMBER:027

TOPIC(S) FOR LECTURE:

1. **High level object-oriented design**
2. **Steps in preliminary design**
3. **Validating preliminary design**
4. **Notation for preliminary design**

INSTRUCTIONAL OBJECTIVE(S):

1. **Understand the steps in preliminary design and how to test it.**
2. **Develop a preliminary object-oriented design.**

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

In our previous discussion, we introduced some of the elements of object-oriented development.

(Learning Label- Today we are going to learn ...)
L27OH1
One can use the preliminary products of structured analysis to aid in the development of object-oriented design. Today we shall examine the components of an object-oriented design and look at the detail of a preliminary object-oriented design.

CONTENTS:
L27OH2

1. Review basic object-oriented concepts, especially message passing and encapsulation since this will get connected with Ada later on. Illustrate these using examples consistent with the KoFF system. Examples of object classes include a video-tape and a VCR. The particular instances could be a particular video tape and your VCR. Message passing can be illustrated by : pushing the display button on the VCR or the end of tape message on the particular video tape which sends a message to the VCR activating a number of processes. The VCR is a good model for explaining encapsulation in so far as we do not know the internal workings of the VCR activated by the PAUSE or DISPLAY buttons. Inheritance is illustrated by talking about the classification of the video tape movies. Each movie has many characteristics in common with other movies but they are separated by classification into 'G', 'PG', etc. This is modeled in a class hierarchy. For example we could have a superclass tape, and each subclass type of type --'G', 'PG', etc -- inherits characteristics from the super class. Each tape could also be considered s an aggregate of 2 take up reels, one tape, and one case.

L27OH3

2. Give an overview of both preliminary and detailed design. Be sure to revisit the concept of domain analysis and other information gathering techniques. Discuss class design as a method to decompose a system into sub-systems and how this can be done in terms of external system responsibilities during preliminary design. The students might find this easier to understand if you talk in terms of architectural responsibilities. The interfaces that are considered at this stage are related to the external behavior required by the user to access the system. Design validation is an important concept to reinforce. The input documents such as event lists, use cases or functional requirements lists can be traced to see that the design meets all of the system requirements. Briefly discuss the outputs of preliminary design with an emphasis on the traceability matrix and the information from preliminary design passed to detailed design. Show the detailed design slide very briefly. Emphasize that this is where implementation details begin to appear. L27OH4

3. After the overview of design, begin a detailed discussion of object identification, as the first step in preliminary design. L27OH5 indicates the different kinds of things that are candidates for objects for a system.

   Ask the students to think of systems where each of these might be an object. You might also use the KoFF system, a garage door opener, or a spell checker. Others have used ATM systems, motor vehicle registration systems , and air traffic control systems. The goal is to get them to think of system objects as including more than tangible things.

   L27OH6

   a. Discuss various object identification techniques.

      i Be sure to admit the limitations of the noun identification technique. Show the KoFF system L27OH7 description and start to list the nouns in it. L27OH8 Then show a partial noun list and ask them to identify synonyms, e.g, membership card and movie rental card. Have them remove the synonyms. Tell them that objects have attributes, and one simple way to identify attributes of objects is to look for things that cannot exist independently but must be properties of something else. For example, color cannot exit alone but must be the property of some object. Similarly, in the KoFF system, Price would be an attribute of a TAPE. Look at the list again for nouns which are attribute candidates, e.g., due-date must be the property of some object. Then have them group some the nouns into major categories, such as billings, member, and tapes. These categories include most of the nouns which can be potential objects or can be major partitions of the design.

ii Discuss use cases as another identification technique. L27OH10 Conversations with the customer generate descriptions of external system behavior which can be modeled in mini-scenarios, called use cases. These scenarios might reveal some additional system objects. (This is a recent technique developed by Jacobsen).

b. Return to the Object identification slide and discuss a method for identifying objects behavior which is based on finding verbs which indicate an objects responsibility. L27OH6 L27OH9 Use the KoFF system description and look for verbs, associating each verb with some object identified in the noun identification pass. These verbs are candidates for object operations. Sometimes operations have characteristics, such as time constraints, or the number of times they can be repeated. These constraints can be indicated by adverbs. Examine each object in the list for relevancy. Remove synonymous objects and objects which are not directly related to the system. Remove nouns which cannot exist independently.

4. Present the Rumbaugh object-model notation as a way to describe the objects that have been identified. Go over the notation. Be sure to discuss: multiplicity, association, generalization and specialization, and subclass and superclass. It is helpful to use examples from their small projects here. L27OH11 L27OH12 L27OH13 For example, aggregation can be illustrated by talking about a vending machine consisting of slots and a change maker. A video tape can also be considered an aggregate of: the tape, two take up reels, and the case. Derived attributes can be illustrated with KoFF tape due date, since it is a function of the rental duration and the initial rental date.

L27OH14

5. Clearly discuss the deliverables required for the preliminary design for the extended project. An object model consisting of an object diagram using Rumbaugh's notation is required. They use OMTool for this. The object model also includes an object dictionary and a traceability matrix. The importance of the object traceability matrix for testing and

further development is emphasized. They are also expected to do Ada specifications for each object as an interface description. The preliminary design team should also develop all user interfaces, e.g., design major menus for the system. (If you have a user interface team, instead of a user manual team, then the design of menus belongs to the user interface team)

L27OH15, L27OH16

6. The Class dictionary provide significant information needed for design and implementation. The developer can use this dictionary to cross check the attributes of this class. The specification of the information needed from other objects helps in the interface design. The object traceability matrix is used to verify that every requirement is accounted for in the object design.

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

<u>vocabulary introduced</u>:

traceability matrix
object class
instance
encapsulation
message
method
inheritance
class hierarchy
object model notation
class dictionary
subclass
generalization
specialization

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>:

L27OH1	Outline
L27OH2	Definitions
L27OH3	Object-oriented design (1)
L27OH4	Object-oriented design (2)
L27OH5	Identification of objects
L27OH6	Object identification techniques
L27OH7	KoFF Automated Video Rental System description
L27OH8	A KoFF partial noun list
L27OH9	A KoFF partial noun and verb list
L27OH10	KoFF use cases

L27OH11	**Examples of Rumbaugh object-oriented design notation**
L27OH12	**More examples of Rumbaugh object-oriented notation**
L27OH13	**Object model notation based on Rumbaugh et al.**
L27OH14	**Preliminary design deliverables**
L27OH15	**Example layout of class dictionary**
L27OH16	**Example object traceability matrix entry**

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 022 - Ada laboratory environment

READING ASSIGNMENTS:
Mynatt Chapter 8 (pp. 364-368)
Sommerville Chapter 10 (pp. 177-182)
Sommerville Chapter 11 (pp. 194-236)

RELATED READINGS:
Ghezzi Chapter 4 (pp. 115-122)
Pressman Chapter 12 (pp. 395-418)
Ivor Jacobson, Object-Oriented Software Engineering, ACM Press
James Rumbaugh, et.al, Object-Oriented Modeling and Design, Prentice Hall

OUTLINE

Definitions

Design

- High Level Design (Preliminary Design)
- Low Level Design (Detailed Design)

Steps in High Level Design

Notation to Express the High Level Design Model

Preliminary Design Deliverables

Definitions

OBJECT CLASS	Models "things" in the world: the model has attributes, operations, and a precise interface that receive messages. (a factory waiting to create instances)
INSTANCE	An actual object waiting to perform services and having state. (the object)
ENCAPSULATION	An object's state data cannot be directly accessed, it can only be asked for a service.
MESSAGE	The only way objects communicate and request services from other objects.

Definitions (cont.)

METHOD	An object class's service or behavior in response to a message.
INHERITANCE	The state and services of a superclass are available to a subclass.
AGGREGATE	An object made up of several components.
CLASS HIERARCHY	

OBJECT-ORIENTED DESIGN (1)

High Level Design

Input	Requirements Documents
	Costumer Interviews
	Domain Analysis
Process	Identify Domain Object Classes
	Class Design
	Divide System Responsibilities
	Design Interfaces
	Identify Object Relationships
	Design Validation
Output	Preliminary design deliverables

OBJECT-ORIENTED DESIGN (2)

Low Level Design

Input	Preliminary Design Deliverables
Process	Identify Internal aspects of Objects
	Data Structure Design
	Algorithms for Operations
	Identify Object Relationships
	Validation
Output	Detailed design deliverables

Identification of Objects

Potential Objects are:

Devices the system interacts with

Events

Incidents

Interactions

Locations of things

Organizations

Remembered Events

Roles of People or Things

Systems outside the current application

Tangible things

Where Have all the Objects Gone?
Object Identification Techniques

First phase:

Name Objects

A noun list from requirements or customer conversations.

Data Dictionary entries*

Data Flow Diagrams*

Requirements List*

Use cases

Determine Object's Behavior

A verb list (Responsibilities)

Assign Methods to Objects

Eliminate Irrelevant Objects

(* Used later in validation of object identification)

Object Identification Techniques (cont.)

Second phase: (using remaining objects)

Assign attributes

Abstract superclass objects based on common behaviors and objects

Distinguish Private methods from public contracts

Automated Video Rental System Description

Client Request

Mr. Richard wants a computerized automated video cassette rental system which will be housed in unmanned kiosks. These kiosks can be free standing in mall parking lots or can be placed in enclosed shopping malls. This device, KoFF (Kiosk of Famous Flicks), will accept applications for membership in Mr. Richard's Rapid Rental club (RRR), display titles of available tapes, dispense tapes, accept returned tapes, and take care of billings. It will also maintain reports of rental transactions.

One becomes a member of the club by entering membership information on a keyboard attached to the kiosk. This information will include a current charge card number and an approval to automatically charge that card for selected items including a membership fee of $ 10.00. Customers will be notified of membership in RRR by mail and will receive three RRR movie rental cards and a unique personal identification number. Membership expires on the expiration date of their charge card.

The kiosk contains 250 different tape titles and 1380 individual tapes. A customer can see a list of the available tapes by category by inserting one of their membership cards into the kiosk. The customer can select an available tape and rental duration. They will be charged for it and the tape will be dispensed from the tape out slot. Their card will be retained until the tape is returned to that kiosk. When a tape is returned to the tape-in slot, its bar code will be scanned, the customer will automatically be charged appropriate late fees and the membership card will be returned. Failure to return the tape within five days of its due date generates a phone call to the customer which plays a recorded message about the overdue tape and the accruing late charges. When the 10-day late limit is reached, the customer is charged for the late days and the cost of the tape. The customer is also charged a tape restocking fee and all of his/her membership cards are invalidated. The customer is notified of these actions.

The selection of videos must be updated. KoFF keeps information to help in this process. Videos which have not been rented for two weeks are listed for removal and videos which have been rented several times in a week are listed for additional copies. Every two weeks KoFF sends Mr. Richard's computer a copy of this report. He decides which tapes to add and which to remove. He updates the list of titles and records the quantities of those titles along with their identifying bar codes. He also assigns the rental price for that title. Sometimes instead of replacing a slow moving tape, he simply drops its rental price or tries to sell it. Sale tapes are indicated on a special screen. When a customer selects a sale tape, a record of the sale is made and the tape is dispensed.

Mr. Richard gets several reports from KoFF, including lists of sold tapes, the rental activity of RRR members by tape title and tape category -- Adventure, Comedy, Children, Restricted, the rental activity of particular titles and copies of that title, and detailed and summary financial reports of RRR member accounts.

A KoFF Partial Noun List

NOUNS	Major Categories
Kiosk	
tapes	tapes
titles	
billings	billings
rental transactions	
member	member
membership information	
keyboard	
charge card number	
membership fee	
movie rental cards	
personal identification number	
expiration date	
list	
membership card (syn)	
available tape	
rental duration	
tape-out slot	
tape-in slot	
bar-code	
due-date	
phone-call	
late-fees	
overdue tape	
etc.	

A KoFF Partial Noun and Verb List

Noun	Categories	Verbs
Kiosk		
tapes	tapes	dispense, accept, report, list
titles		
billings	billings	
rental transactions		
member	member	enter info, charge fee,
membership information		issue card
membership fees		

KoFF USE CASES

Rent a Tape

Buy a Tape

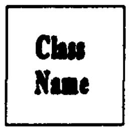
Return a Tape

Membership Card Rejection

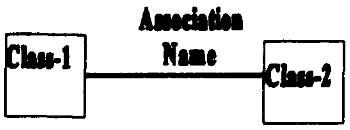
Rent a tape:
Conversation with customer:"How do you want someone to rent a tape?"

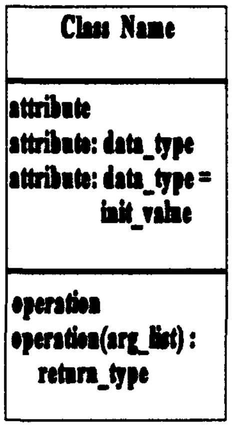
What new information does this reveal?
"What are the desired functions for tape rental?"

Examples of Rumbaugh Object-Oriented Design Notation

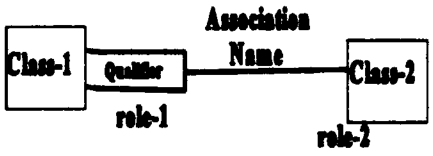


Association:



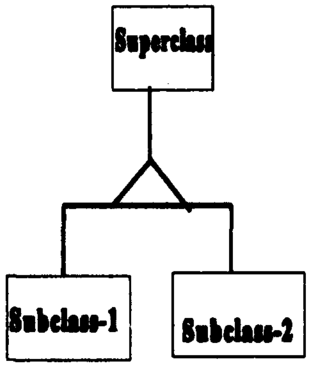

Class Name
attribute
attribute: data_type
attribute: data_type = init_value
operation
operation(arg_list) : return_type

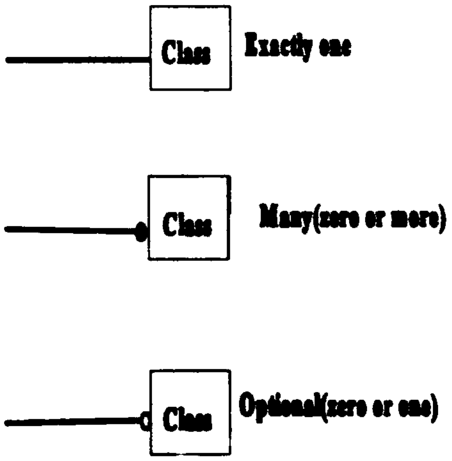
Qualified Association:



Multiplicity of Association:

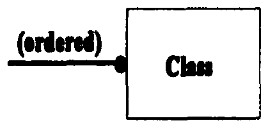
Generalization(Inheritance):



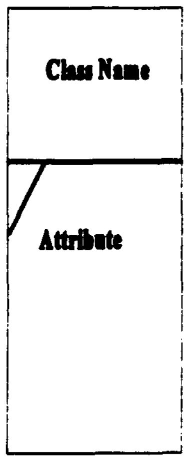


More Examples of Rumbaugh Object-Oriented Design Notation

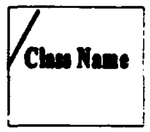
Ordering:



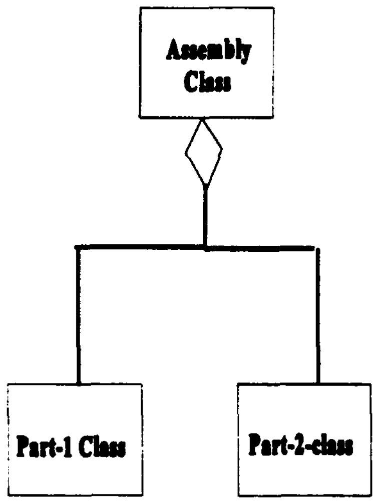
Derived Attribute:



Derived Class:



Aggregation:



KoFF USE CASES

Rent a Tape

Buy a Tape

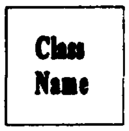
Return a Tape

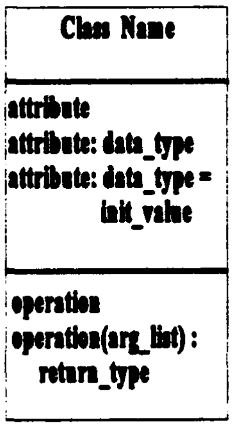
Membership Card Rejection

Rent a tape:
    Conversation with customer:"How do you want someone to rent a tape?"

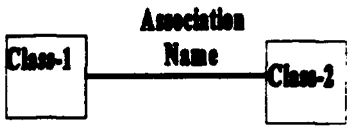
What new information does this reveal?
    "What are the desired functions for tape rental?"

Examples of Rumbaugh Object-Oriented Design Notation

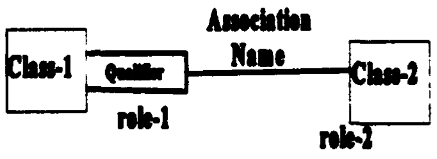



Class Name
attribute
attribute: data_type
attribute: data_type = init_value
operation
operation(arg_list) : return_type

Association:

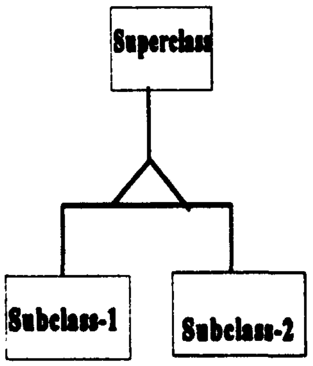


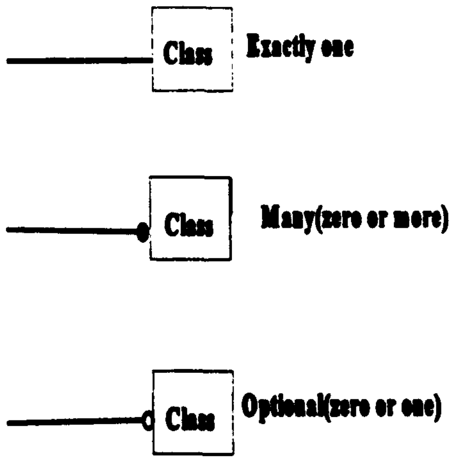
Qualified Association:



Multiplicity of Association:

Generalization(Inheritance):





Object Model Notation
Based on Rumbaugh et al.

Aggregation:

A special form of association between a whole and its parts.

Association:

A relationship among instances of two or more classes.

Association as a class:

Each link is an instance of a class.

Qualifier:

Reduces the multiplicity of an association at the many end.

Role:

Appear as nouns in product description and uniquely identify one end of an association.

Preliminary Design Deliverables

An Object Model:

> A complete object diagram using Rumbaugh notation as presented in class.
>
> A Class Dictionary entry for each object.
>
> An Object-Requirements traceability matrix.

Ada Specifications for each object class.

Descriptions of all major user interfaces, e.g., menu formats and options.

CLASS DICTIONARY

OBJECT CLASS NAME:

OBJECT DESCRIPTION:

ATTRIBUTE DESCRIPTION:

-
-
-
-

Method Descriptions:

-
-
-
-

Input information needed from other objects:

-
-
-
-

Object Traceability Matrix

Functional Requirement ID	Object Name	Ada Package
1	MEMBER	MEMBERSHIP_OPS

LECTURE NUMBER: 028

TOPIC(S) FOR LECTURE:
Ada packages

INSTRUCTIONAL OBJECTIVE(S):

1. To understand the use and usefulness of Ada packages.
2. To learn the syntax of Ada packages.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

Soon the students will be working on the preliminary design of their second project which is to be implemented in Ada. The preliminary design is also to be accomplished using Ada package specifications. The students have been introduced to Ada packages in previous classes.

(Learning Label- Today we are going to learn ...)

In today's lecture, they will have a more detailed look at the syntax of Ada packages through several examples.

CONTENTS:

L28OH1

1. Give a brief overview of three program units in Ada - procedures and functions, packages, and tasks. Relate procedure and functions to a language students are already familiar with. The development of packages is a primary goal of the preliminary design team. Be sure to emphasize the correspondence between packages and objects.

2. Detailed look at Ada packages L28OH2

   a. A package is a collection of related entities available for use by other program units. A package can include constants, variables, types, procedures, functions, exceptions, tasks, and even other packages. Packages are passive. They have to be invoked by operations of other entities.

   b. A package can be used for a collection of declarations, a group of related program units, or an abstract data type.

   c. There are two parts to a package. They are stored as two different files, each having the same name but a different extension:

i The specification is the public or visible part which provides the interface information and indicates the entities which are made available by this package. The public or vissible part is what you are going to provide to the system. This corresponds to OMT's method part of an object. The specification can be created in the preliminary design phase of the software life cycle.

ii The body is the hidden part which contains the implementation details. Knowledge of the package details of the package body is not needed in order to use the package. The package body contains the bodies of the subprograms which are declared in the package specification. If also contains local declarations, and subprograms which are inaccessible to the user of the package. The package body is optional. Sometimes it is not needed; for example, a package may contain only declarations such as shown in L28OH3.

d. Discuss/review the software engineering concepts of abstraction, encapsulation, information hiding, modularity, and reusability and how these are supported by Ada packages as in L28OH4.

e. Ada supports various levels of information hiding and encapsulation. What follows is a series of refinements of an Ada Queue package, each of which more effectively hides and encapsulates information.

i. L28OH5 Presents a package in which all data and operations are publicly accessible and the body L28OH6 has minimal details. A Procedure X could use a Queue_type, but it could only Enter and Remove. (E.G. procedure X is

```
        trans A
        QueueA, QueueB:Queue_type;

        enter(TransA,QueueA);
```

ii. L28OH7 Declares the Queue type as a private type, and hides implementation details such as the fact that it is a linked list.

iii. L28OH8 Here a greater degree of encapsulation is achieved by moving all implementation details to the package body. L28OH9. All that is visible is the interface transaction type and the two procedures,Enter and Remove. Data has to be accessed by the method specified in the package specification.

f. L28OH11 - If in, out, or in out, is omitted from a procedure or function header, the parameter defaults to an in parameter. In the formal parameter list for a procedure or function each parameter is labeled as 'in' (for an input parameter), 'out' (for an output parameter, and 'in out". The parameter defaults to in.

PROCEDURE:

teaching method and media:

Lecture and overheads are the chief media for this lecture.

vocabulary introduced:
Ada packages
Ada package specification
Ada package body
private data type
limited private data type

INSTRUCTIONAL MATERIALS:

overheads:

L28OH1	Ada program units
L28OH2	Ada packages
L28OH3	Example of package specification - Solar System
L28OH4	Software engineering concepts supported by Ada packages
L28OH5	Example of package specification - Queue
L28OH6	Example of package body - Queue
L28OH7	Example of package specification with data declarations in private section - Queue
L28OH8	Example of package body to go with overhead 7
L28OH9	Example of package specification with all data inside package body.
L28OH10	Example of package body to go with overhead 9
L28OH11	Procedure and function headers

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 023 Peer reviews and preliminary design review presentaiton

READING ASSIGNMENTS:

Benjamin Chapter 8 (pp. 73-78)
Sommerville Appendix(pp. 610-613)

RELATED READINGS:

Booch Chapter 6 (pp. 53-74)
Booch(2) Chapter 4 (pp. 43-65)

Ada Program Units

Procedures and functions

Packages

Tasks
    Provides concurrency

Ada Packages

A program unit that allows a collection of related entities to be made available for use by other program units

Two parts to a package:

Specification

The public or visible part
Interface information

Body

The hidden part
Implementation details

Software Engineering Concepts Supported by Ada Packages

Abstraction
: A view of a problem that extracts the essential information relevant to a particular purpose and ignores the remainder of the information

Encapsulation
: The technique of isolating data and related procedures/functions within a module and providing a precise specification for the module

Information hiding
: The technique of forbidding the use of information about a module that is not in the module's interface specification

Modularity
: The purposeful structuring of the modules of a system so that a change to one component has minimal impact on other components

Reusability
: The extent to which a module can be used in multiple applications

Example of Package Specification

```
package SOLAR_SYSTEM is

 type PLANET is (MERCURY, VENUS, EARTH,
     MARS,  JUPITER,  SATURN,  URANUS,
     NEPTUNE, PLUTO);

 subtype TERRESTRIAL_PLANET is PLANET
     range MERCURY..MARS;

 NUMBER_OF_MOONS:constant array (PLANET)
     of NATURAL := (MERCURY => 0, VENUS =>
     0, EARTH => 1, MARS => 2, JUPITER => 12,
     SATURN => 10, URANUS => 5, NEPTUNE
     => 2, PLUTO => 0);

end SOLAR_SYSTEM;
```

Example of Package Specification

```
package QUEUE is
 type TRANSACTION is
   record
     ACCOUNT_ID : integer;
     NAME : string;
     ADDRESS : string;
   end record;
 SIZE : constant POSITIVE := 10;
 subtype INDEX is integer range 1..SIZE;
 type SPACE is array (INDEX) of TRANSACTION;
 type QUEUE_TYPE is
   record
        ITEMS : SPACE;
       HEAD  : INDEX := 1;
       TAIL  : INDEX := 1;
       COUNT : integer range 0..SIZE := 0;
   end record;

  procedure ENTER (T : in TRANSACTION;
                  Q : in out QUEUE_TYPE);
  procedure REMOVE (T : out TRANSACTION;
                   Q : in out QUEUE_TYPE);
end QUEUE;
```

Example of Package Body

```
package body QUEUE is

  procedure ENTER (T : in TRANSACTION;
                   Q : in out QUEUE_TYPE);
  begin
     ...
  end;

  procedure REMOVE (T : out TRANSACTION;
                    Q : in out QUEUE_TYPE);
  begin
     ...
  end;
end QUEUE;
```

Example of Package Specification

```
package QUEUE is
  type TRANSACTION is
     record
        ACCOUNT_ID : integer;
        NAME : string;
        ADDRESS : string;
     end record;
  type QUEUE_TYPE is private;
  procedure ENTER (T : in TRANSACTION;
                  Q : in out QUEUE_TYPE);
  procedure REMOVE (T : out TRANSACTION;
                   Q : in out QUEUE_TYPE);
  private
    SIZE : constant POSITIVE := 10;
    subtype INDEX is integer range 1..SIZE;
          type SPACE is array (INDEX) of
TRANSACTION;
    type QUEUE_TYPE is
       record
         ITEMS : SPACE;
         HEAD  : INDEX := 1;
         TAIL  : INDEX := 1;
         COUNT : integer range 0..SIZE := 0;
       end record;
end QUEUE;
```

Example of Package Body

```
package body QUEUE is

  procedure ENTER (T : in TRANSACTION;
                   Q : in out QUEUE_TYPE);
  begin
     ...
  end;

  procedure REMOVE (T : out TRANSACTION;
                    Q : in out QUEUE_TYPE);
  begin
     ...
  end;
end QUEUE;
```

Example of Package Specification

```
package QUEUE is

 type TRANSACTION is
    record
       ACCOUNT_ID : integer;
       NAME : string;
       ADDRESS : string;
    end record;

 procedure ENTER (T : in TRANSACTION);

 procedure REMOVE (T : out TRANSACTION);

end QUEUE;
```

Example of Package Body

```
package body QUEUE is
   SIZE : constant POSITIVE := 10;
   subtype INDEX is integer range 1..SIZE;
          type   SPACE   is   array   (INDEX)   of
TRANSACTION;
   type QUEUE_TYPE is
      record
         ITEMS : SPACE;
         HEAD  : INDEX := 1;
         TAIL  : INDEX := 1;
         COUNT : integer range 0..SIZE := 0;
      end record;

   A_QUEUE : QUEUE_TYPE;

  procedure ENTER (T : in TRANSACTION) is
  begin
     ...
  end;

  procedure REMOVE (T : out TRANSACTION) is
  begin
     ...
  end;
end QUEUE;
```

Procedure and Function Headers

```
procedure ENTER (T : in TRANSACTION);


procedure REMOVE (T : out TRANSACTION);


procedure MODIFY (T : in out TRANSACTION);


function CUBE (X : integer) return integer;
```

LECTURE NUMBER: 029

TOPIC(S) FOR LECTURE:

1. Introduction to software quality assurance (SQA)
2. Reviews - walkthroughs and inspections

INSTRUCTIONAL OBJECTIVE(S):

1. Understand the scope of quality assurance and the related activities.
2. Understand the purpose of technical reviews, specifically of walkthroughs and inspections.
3. Understand general guidelines for technical reviews.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

The issue of software quality has come up in a number of earlier lectures, both directly and indirectly. Recall, for example, verification and validation (V&V) - see L29OH1. V&V activities were aimed at ensuring two things:

1) verifying that the system meets its specification (Are we building the product right?); and

2) validating that system as implemented meets the clients'/users' expectations (Are we building the right product right?).

V&V are activities undertaken to increase the chances of achieving software quality. Similarly, recall configuration management deals with controlling and managing change. CM activities are also undertaken in order to increase the chances of achieving software quality. Both V&V and CM activities are software quality assurance (SQA) activities.

(Learning Label- Today we are going to learn ...)

Today we're going to look further at SQA.

CONTENTS:

1. L29OH2
   Pressman defines SQA as an "umbrella activity" which encompasses V&V, CM and a number of other types of activities. SQA is concerned with both product and process quality and it encompasses:

   a. Technical methods and tools - These are used throughout the software life cycle. They include methods and tools to aid in developing high-quality specifications, to methods and tools for implementation and testing.

b. Technical reviews - These are also applied at every stage of the software life cycle.

c. Testing - Testing is used to reveal the presence of problems at any stage of developement.

d. Configuration management - The control and management of change applied to all artifacts of software development.

e. Standards - Both the development of and compliance with standards.

f. Measurement and reporting - These activities involve measurement to track quality and to improve quality by modifying the process in light of these measurements.

Use L29OH3 and L29OH4 to explain the importance of SQA at the beginning of the development process.

2. L29OH5 What is software quality?
Pressman defines software quality as "conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

Point out that there are three components of quality:

a. Meets (explicit) requirements; (This involves the software product).

b. Meets (explicit) standards; (This involves the software process.) The quality of the process affects the quality of the product.

c. Meets (implicit) requirements - these are standards expected of all professionally developed software; Note that software could meet all of its explicitly stated requirements yet still be of questionable quality. These are some things that a customer should expect even without saying so.

Discuss some of these implicit characteristics that should be expected of all quality software? (Let short discussion bring out such things as maintainability, reusability, robustness, portability, etc).

L29OH6 shows how quality factors are evidenced in the final product.

3. Software reviews - Reviews are one of the most important SQA activities.

L29OH7
A review of some component of a software development process serves to uncover defects so that they can be corrected/removed before going on. There are many types of reviews, both formal and informal.

a. The most common types of reviews are called structured walkthroughs and inspections. In both, a team of software professionals carefully reviews an item, thereby increasing the chances of defects being located.

b. L29OH8
Technical reviews uncover errors in the item under review, verify that the software under review meets its requirements, and ensure conformance to standards. Note also that reviews can serve as training activities to new and/or inexperienced personnel.

c. Reviews occur at meetings. Guidelines for review meetings typically suggest participation of 3-6 appropriate people and require advance preparation of 1-2 hours. The focus of the review is the improvement of the product under review.

L29OH9
Roles for a review include:

i Review leader - evaluates review item for readiness, distributes review materials to reviewers, typically a day before the review. The review leader, like the other reviewers, is expected to spend 1 to 2 hours reviewing the material in preparation for the review. The review leader also schedules the review and prepares the agenda.

ii Recorder - One of the reviewers is responsible for recording (in writing) all important issues raised during the review.

iii Producer - The developer of the item under review "walks through" the product, explaining the material, while reviewers raise issues identified in their advance preparation or during the review itself.

iv Other reviewers - Each has carefully reviewed the materials and comes prepared with a list of items not understood and a list of items he/she believes to be incorrect.

d. There are different methods of conducting the review. One is "participant-driven", in which each participant goes through his/her lists of unclear/incorrect items and the presenter responds. Another is "document-driven" in which the presenter(s) walk the reviewers through the item under review, and the reviewers bring up their concerns as they are encountered. The document-driven approach is more thorough. In practice, a majority of faults in document driven walkthroughs are detected by the presenter during the walkthrough.

4. L29OH10 Review reports
The recorder notes all issues raised that need to be addressed. These are summarized at the end of the review and a "review issues list" is produced. A "review summary report" is also completed, containing the item reviewed, names of reviewers, and findings and conclusions.

Discuss examples in in L29OH11 and L29OH12 The review issues list identifies problem areas and serves as action item checklist for the producer as he/she addresses the issues.

5. L29OH13
Discuss the review guidelines, adapted from Pressman.

a. Review the product, not the producer.

b. Set and maintain an agenda.

c. Limit debate and rebuttal.

d. Focus on identifying problems, not on attempting to solve them.

e. Keep a written record.

f. Limit the number of participants.

g. Insist upon advance preparation of participants.

h. Develop a checklist for likely review items.

i. Allocate resources for reviews.

j. Provide appropriate training for reviewers.

k. Establish a follow-up procedure to assure that items on review issues list are addressed.

I. Do not allow reviews to be used as a means of assessing participants.

6. Effectiveness of reviews - Evidence has shown that formal technical reviews are extremely effective in meeting their objectives.

PROCEDURE:

teaching method and media:

vocabulary introduced:

quality
implicit requirements
explicit requirements
software quality assurance (SQA)
technical reviews
walkthrough
inspections

INSTRUCTIONAL MATERIALS:

overheads:

L29OH1	V & V activities
L29OH2	Software quality assurance
L29OH3	Source of errors by life cycle phase
L29OH4	Relative cost of errors by life cycle phase
L29OH5	Software quality
L29OH6	McCall's software quality factors
L29OH7	Software reviews
L29OH8	Purpose of technical reviews
L29OH9	Review roles
L29OH10	Review reports
L29OH11	Review issues list
L29OH12	Technical review summary report
L29OH13	Review guidelines

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 024 - User interface presentation/review
Test plan presentation/review

READING ASSIGNMENTS:

Sommerville Chapter 31 (pp. 589-598)
Mynatt Chapter 2 (pp. 77-79)

RELATED READINGS:

Pressman Chapter 17 (pp. 549-589)
Schach Chapter 5 (pp. 101-109)

Relationship of V&V and CM to SQA

V&V activities aimed at:

> Verifying that the system meets its specification (Are we building the product right?); and
>
> Validating that system as implemented meets the clients'/users' expectations (Are we building the right product right?).

CM activities aimed at:

> Controlling change and
>
> Managing change.

V&V and CM activities are both undertaken in order to increase the chances of achieving software quality.

Both are software quality assurance (SQA) activities.

Software Quality Assurance (SQA)

SQA is an "umbrella activity" which encompasses V&V, CM and a number of other types of activities. SQA is concerned with both product and process quality.

SQA encompasses:

Technical methods and tools

Technical reviews

Testing

Configuration management

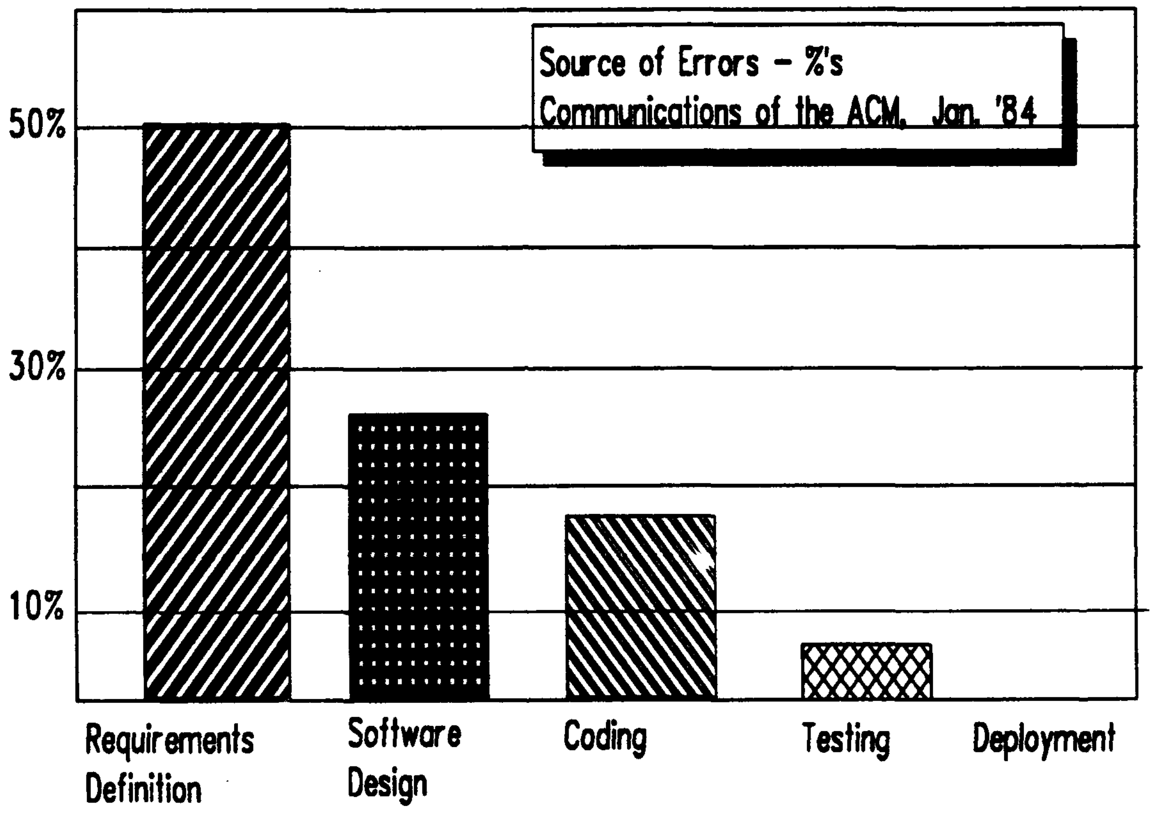
Standards

Measurement and reporting

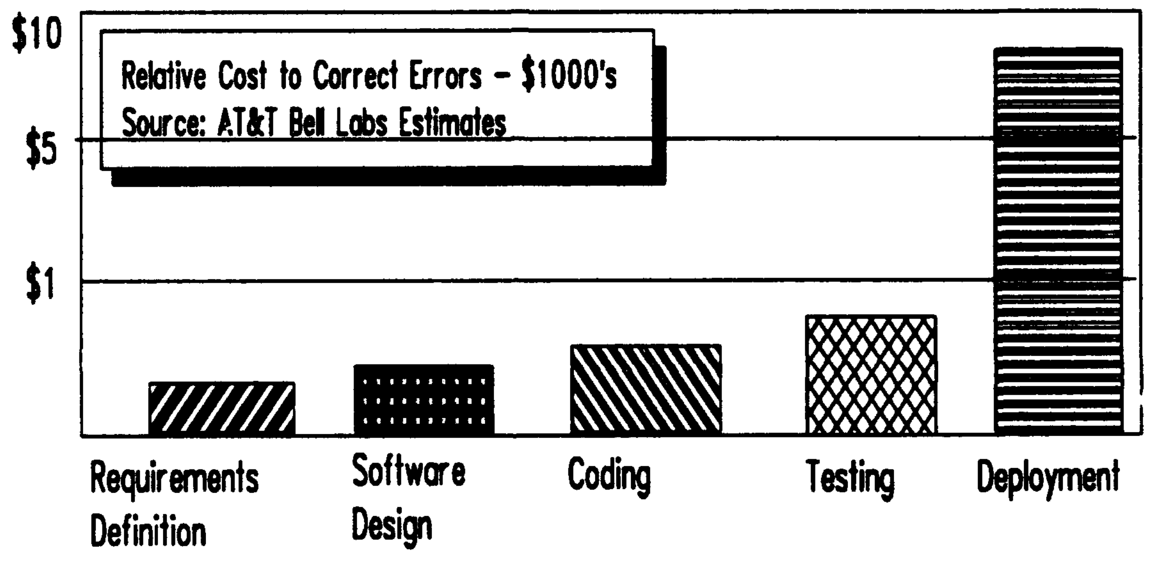
SQA is concerned with "whole-life cycle" quality.

Source of Errors by Life Cycle Phase

Errors Found Early are Easier
To Find and Manage



Relative Cost of Errors by Life Cycle Phase



Software Quality

"conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software".

source: Pressman

Three components of software quality

Meets (explicit) requirements

Meets (explicit) standards

Meets (implicit) requirements

McCall's Software Quality Factors

Maintainability (Can I Fix it?)

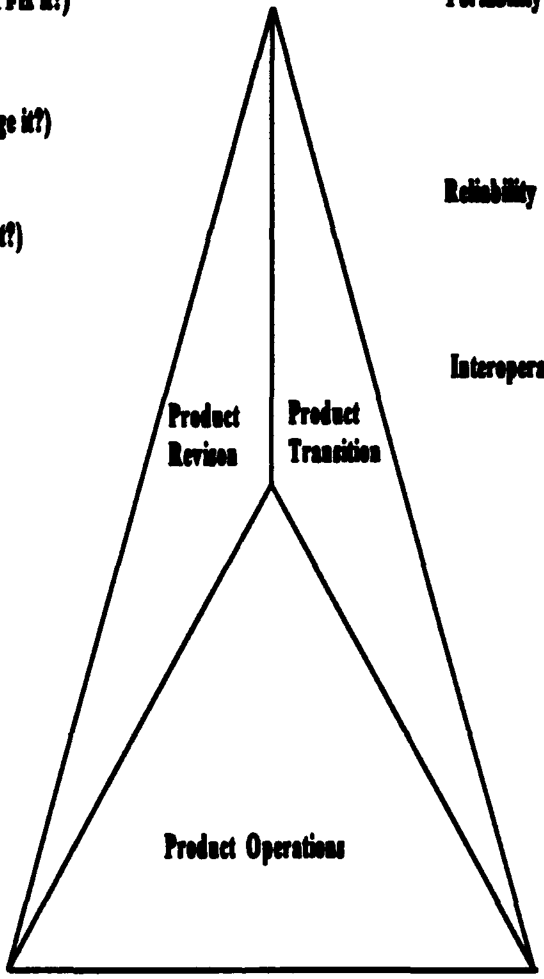
Flexibility (Can I change it?)

Testability (Can I test it?)

Portability (Will I be able to use it on another machine?)

Reliability (Will I able to reuse some of the software?)

Interoperability (Will I be able to interface it with another system?)



Correctness	(Does it do what I want?)
Reliability	(Does it do it accurately all of the time?)
Efficiency	(Will it run on my hardware as well as it can?)
Usability	(Can I run it?)

Software Reviews

One of the most important SQA activities.

A review is intended to uncover defects so that they can be corrected/removed before going on.

Two common types of reviews:

Walkthroughs

Inspections

Both involve a group of software professionals carefully reviewing an item, thereby increasing the chances of defects being located.

Purpose of Technical Reviews

To uncover errors in the item under review.

To verify that the software under review meets its requirements.

To ensure conformance to standards.

Reviews can also serve as training activities to new and/or inexperienced personnel.

Review Roles

Review leader

Evaluates item for readiness

Distributes review materials in advance

Reviews the material prior to meeting

Schedules review and prepares agenda

Recorder

Records all important issues raised during review

Producer

"Walks through" the product, explaining the material, while reviewers raise issues based on their advance preparation

Other reviewers

Review materials in advance and come prepared with a list of items not understood and a list of items he/she believes to be incorrect

Review Reports

Review issues list

Identifies problem areas raised during review that need to be addressed

Serves as action item checklist for the producer as he/she addresses the issues

Review summary report

Item reviewed;

Reviewers;

Findings and conclusions.

Review Issues List

Review Number : D-004
Date of Review: 07-11-86
Review Leader : R.S. Pressman
Recorder : A.D. Dickerson

Issues List

1. Prologues for module YMOTION, ZMOTION are not consistent with design standards. Purpose of the module should be explicitly stated (reference is not acceptable) and data item declaration must be specified.

2. Loop counter for interpolation in X, Y, Z axes increments one time too many for step motor control. Review team recommends a recheck of stepping motor specifications and correction (as required) of the loop counter STEP.MOTOR.CTR.

3. Typo in reference to current X position, X.POSITION in modules XMOTION and ZMOTION. See marked PDL for specifics.

4. PDL psuedo-code statement must be expanded. The psuedo-code statement: "Converge on proper control position as in XMOTION" contained in modules YMOTION and ZMOTION should be expanded to specifics for Y and Z motion control.

5. Review team recommends a modification to the "position comparator" algorithm to improve run time performance. Necessary modifications are noted in annotated PDL. Designer has reservations about the modification and will analyze potential impact before implementing change.

Figure 17.6b - Pressman

Technical Review Summary Report

<u>Review Identification:</u>

Project: Review Number:
Date: Location: Time:

<u>Product Identification:</u>

Material Reviewed:

Producer:

Brief Description:

<u>Material Reviewed:</u> (note each item separately)

<u>Review Team:</u> (indicate leader and controller)

	Name	Signature
1.		
2.		
3.		
4.		

<u>Product Appraisal:</u>

Accepted: as is ( ) with minor modification ( )
Not Accepted: major revision ( ) minor revision ( )
Review Not Completed: (explanation follows)

<u>Supplementary Material Attached:</u>

Issues List ( ) Annotated Produce Materials ( )
Other (describe)

Figure 17.6b - Pressman

Review guidelines

Review the product, not the producer

Set and maintain an agenda

Limit debate and rebuttal

Focus on identifying problems, not on attempting to solve them

Keep a written record.

Limit the number of participants.

Insist upon advance preparation of participants

Develop a checklist for likely review items.

Allocate resources for reviews.

Provide appropriate training for reviewers.

Establish a follow-up procedure to assure that items on review issues list are addressed.

Do not allow reviews to be used as a means of assessing participants.

LECTURE NUMBER: 030

TOPIC(S) FOR LECTURE:

Review standards and checklists

INSTRUCTIONAL OBJECTIVE(S):

1. Understand that review standards exist for various software life cycle stages and products.
2. Become familiar with some review standards.
3. Become familiar with the concept of review checklists, particularly for preliminary design reviews and detailed design reviews.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

We recently talked about technical reviews (walkthroughs and inspections) as a primary SQA activity.

(Learning Label- Today we are going to learn ...)

Today we want to introduce you to some accepted standards for reviews.

CONTENTS:

1. L3OH1
   Many professional and governmental organizations have developed standards for SQA. For example, the DOD has an SQA standard which we have discussed earlier (DOD-STD-2167A) and which covers the entire software development life cycle. Other government agencies with unique requirements, such as the Federal Aviation Administration (FAA), have their own standards.

   L30OH2
   The IEEE has a standards development organization which builds quality standards for many of the phases of software development. They have, in fact, developed a general SQA plan. Discuss aspects of the IEEE SQA Plan.

2. L30OH3
   In order to achieve SQA, reviews must be conducted at critical points in the software development process. Discuss the critical review points shown.

3. Checklists have been developed for each of the critical reviews. Such checklists help to structure the reviews and assure that important

points are not overlooked. Discuss examples of these checklists.

L30OH4a - Checklist for preliminary design review
L30OH4a - Checklist for detailed design walkthrough
L30OH5a - Checklist for code review - Myers
L30OH5b - continuation of code review checklist - Myers
L30OH6 - Preliminary design review form
L30OH7a - Detailed design review form
L30OH7b - continuation of detailed design review form

PROCEDURE:

teaching method and media:

vocabulary introduced:
review standards
review check lists
critical design review
system test review

INSTRUCTIONAL MATERIALS:

overheads:

L30OH1	IEEE SQA plan - Pressman, p 588
L30OH2	SQA standards - Pressman, p 589
L30OH3	Some important review points
L30OH4	Design review checklist - Pressman
L30OH5	Code review checklist
L30OH6	Preliminary design review checklist
L30OH7	Detailed design review checklist

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

RELATED READINGS:
Pressman Chapter 17 (pp. 586-590)
Meyers (The Art of Software Testing)

SQA Standards

DOD-STD-2167A	Software engineering
DOD-STD-2168	Software quality evaluation standard
FAA-STD-018	SQA standard for the FAA
IEEE Std. 730-1984	SQA plans
IEEE Std. 983-1986	Software quality assurance planning
IEEE Std. 1028-1988	Software reviews and audits
IEEE Std. 1012-1986	Software verification and validation plans

ANSI/IEEE Standards 730-1984 and 983-1986 Software Quality Assurance Plan

I. Purpose of the plan

II. References

III. Management
   A. Organization
   B. Tasks
   C. Responsibilities

IV. Documentation
   A. Purpose
   B. Required software engineering documents
   C. Other documents

V. Standards, practices, and conventions
   A. Purpose
   B. Conventions

VI. Reviews and audits
   A. Purpose
   B. Review requirements
      1. Software requirements review
      2. Design reviews
      3. Software verification and validation reviews
      4. Functional audit
      5. Physical audit
      6. In-process audits
      7. Management reviews

VII. Software configuration management

VIII. Problem reporting and corrective action

IX. Tools, techniques, and methodologies

X. Code control

XI. Media control

XII. Supplier control

XIII. Records collection, maintenance, and retention

Some Important Review Points

Review	Purpose
Systems requirements review	Understand system and interface specifications. Establish major functional baseline.
Software requirements review	Assess the functional requirements and initiate preliminary design.
Master test plan review	Assess initial master test plan, particularly the overall test strategy.
Preliminary design review	Assess the architectural design. Assess the acceptance and system test specs. Establish preliminary design baseline.
Critical design review	Assess detailed design including data base design. Assess final master test plan, integration and unit test specs, and acceptance test procedures. Authorize start of coding.
Code reviews	Assess units of code. Establish test baseline.
System test review	Assess systems test results. Determine readiness for acceptance testing.
Acceptance test review	Assess acceptance test results. Accept software package. Create product baseline and approve operational implementation.

Design Review Checklists - Pressman

Preliminary design review:

1. Are software requirements reflected in the software architecture?
2. Is effective modularity achieved? Are modules functionally independent?
3. Is the program architecture factored?
4. Are interfaces defined for modules and external system elements?
5. Is the data structure consistent with the information domain?
6. Is the data structure consistent with software requirements?
7. Has maintainability been considered?
8. Have quality factors been explicitly assessed.

Design Review Checklists - Pressman

Detailed Design walkthrough

1. Does the algorithm accomplish the desired function?
2. Is the algorithm logically correct?
3. Is the interface consistent with architectural design?
4. Is the logical complexity reasonable?
5. Have local error handling and "antibugging" been specified?
6. Are local data structures properly defined?

Code Review Checklists (Glenford Meyers)

Data reference

1. Unset variables used?
2. Subscripts within bounds?
3. Non-integer subscripts?
4. Dangling references?
5. Correct attributes when aliasing?
6. Record and structure attributes match?
7. Computing addresses of bit-strings?
8. Passing bit-string arguments?
9. Based storage attributes correct?
10. Structure definitions match across procedures?
11. String limits exceeded?
12. Off-by-one errors in indexing or subscripting operations?

Data declaration

1. All variables declared?
2. Default attributes understood?
3. Arrays and strings initialized properly?
4. Correct lengths, types, and storage classes assigned?
5. Initialization consistent with storage class?
6. Any variables with similar names?

Code Review Checklists

Computation

1. Computations on non-arithmetic variables?
2. Mixed-mode computations?
3. Computations on variables of different lengths?
4. Target size less than size of assigned value?
5. Intermediate result overflow?
6. Division by zero?
7. Base-2 inaccuracies?
8. Variable's value outside of meaningful range?
9. Operator precedence understood?
10. Integer divisions correct?

Comparison

1. Comparisons between inconsistent variables?
2. Mixed-mode comparisons?
3. Comparison relationships correct?
4. Boolean expressions correct?
5. Comparison and boolean expressions mixed?
6. Comparisons of base-2 fractional values?
7. Operator precedence understood?
8. Compiler evaluation of boolean expressions understood.

Preliminary Design Review Form

Project Name____________________________

Reviewer Name____________________________

I. High Level Issues

A. Requirements: any requirements missed, requirements over-worked?

B. Design : suggestions for improvement of architecture or procedures; other strategies

II. Design Deliverable Details

A. Test Plan: items over-tested or under-tested, suggested tests

B. DFD: good use of notation, clear model, suggested improvements

C. Comments on other deliverables

Detailed Design Review Form

Project Name :____________________________

Reviewer Name :___________________________

I. High Level Issues

A: Requirements: any requirements missed, requirements over-worked?

B: Design : suggestions for improvement of architecture or procedures; other strategies

C: The Design fits the whole specification including quality standards such as flexibility, friendliness, efficiency, and cost effective.

II Design Deliverable Details

A: Revised Test Plan: items over tested or under-tested, suggested tests

B: Design Model: good use of notation, clear model, suggested improvements

III Detailed Design

A: Can design be implemented easily: availability of adequate programming and testing manpower. Adequate hardware facilities-computer, data storage...

B: Is the design programmable- does not require exotic functions

C: Is there a suggested or obvious order of implementation or approximate times for the development of and description of the production relations between the modules. What is the order of need for equipment required to implement the design.

D: Comments on other deliverables

LECTURE NUMBER: 031

TOPIC(S) FOR LECTURE:

The relation between detailed and high level design.
Detailed design procedures.
Detailed design deliverables.

INSTRUCTIONAL OBJECTIVE(S):

1. Develop a detailed design.
2. Understand different products of detailed design

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

Last time, when we spoke of design, we looked at the general elements of object-oriented design and paid particular attention to the elements of preliminary design. One of the questions we addressed in structured design was "How does one develop a design from the analysis documents." Today we will look at a similar question for object-oriented development.

(Learning Label- Today we are going to learn ...)

How does one move from preliminary object-oriented design to detailed object-oriented design and what are the products of detailed object-oriented design?

CONTENTS:

1. Design consists of several steps L31OH1

L31OH2

2. Briefly remind them of the elements that make up objects. Show them a sample notation from the KoFF System on the board using Rumbaugh notation. For example the subclasses of tapes_for_sale and tapes_to_rent inherit all class attributes from the superclass TAPE. L31OH3 show the notation for inheritance and an association between the Member object class and the Tape object class.

3. Revisit the stages of preliminary design and discuss some sample products of preliminary design. L31OH4

   a. Talk about the Object Traceability Matrix and how it functions to validate the design. It also helps to show the completeness of the design. L31OH5

   Verification matrices tie Ada software components to the deliverables of the previous development phases (i.e., preliminary and detailed design). These verification matrices provide a means of tracing the transitions between all phases of the life cycle. For example, by means of a verification matrix for preliminary design, every Ada package specification can be traced back to an object in the object model, and that object, in turn, can be traced back to the requirement(s) which it satisfies. In this way, verification matrices make visible the relationship of Ada to software analysis and design.

b. When the preliminary design is complete it should be compared to the functional requirements list for consistency and completeness. Each requirement must be satisfied in the design and each design element must be tied to a requirement.

L31OH6

c. Discuss the contents and function of the Class Dictionary and how it provides traceability to the Ada specification for each object. It is through the dictionary that an Ada specification is tied to an object and the object is tied to a requirement through the traceability matrix.

d. Point out that reusability is actually improved because the Class Dictionary does not include:
   i implementation decisions
   ii data type specifications
   iii how operations accomplish their tasks
   iv where an object's methods are called from

e. Similarly Ada specifications provide interface information but they should not provide any of the items in d. above. This information should be put in the body of the package.

4. Detailed design is the selection, specification, design and representation of the internal aspects of objects. Show the detailed design overhead L31OH10. Make clear that detailed design is not intended to change any object interface and therefore should not alter the preliminary design. Since implementation details were not shown in preliminary design's Ada specifications or object model, detailed design should not impact preliminary design. L31OH7 Visible data structures are declared in the package specification and hidden data structures are declared in the body. Visible data structures characterize the data to be passed to other independent packages while the hidden data structures characterize variables used in the formulation of internal operations. This may include defining internal

values of variables. This is a good example of modularization process.

5. The design of the data structures at this stage is significant for later ease and quality of the implementation. Discuss some standards of data structure design. When discussing coupling and cohesion as a standard, be sure to point out that there is coupling and cohesion within an object and between objects.

6. The data structure design is documented in the data dictionary. Show the Data Structure Dictionary and discuss how to fill it out. L31OH8 The algorithms for each method will also be specified in some pseudocode, such as Nassi Shneiderman models.

7. Show the Detailed Design Traceability Matrix and discuss how to fill it out. L31OH9

   To establish traceability between software development phases, we have also designed an object traceability matrix (Figure 4). This matrix provides a backward trace from each design object to a specific requirement (preliminary design traced to requirements) and a forward trace from each design object to a specific Ada specification (preliminary design traced to detailed design). The introduction of this form allows us to revisit the concept of traceability in the software life cycle.

   Traceability is extended into detailed design by means of a detailed design traceability matrix. We created this matrix to provide traceability between preliminary design, detailed design and implementation. The detailed design matrix first provides traceability between an object's attributes and its data structures and between those data structures and their Ada package representation. The matrix also provides traceability between the object's operations, the detailed design model of those operations and the Ada package embodying those operations.

8. Detailed design should also expand upon the user interface developed in preliminary design. Suppose for example the preliminary design team specified the format for the screen used to call help. The type of content needed by each help screen should be determined in detailed design, e.g., will help screens have an option to do a core dump or merely an option to abort or continue processing.

L31OH10,L31OH11

9. These are the deliverables of preliminary and detailed design. There may be some continuing problems which need to be addressed. There is a possibility that this low level design will reveal problems with the earlier stages of the software development. Problems could include missing, contradictory or non-feasible requirements. They could also include requirements which were not satisfied by the preliminary design. Before any changes can be made, configuration management is alerted. The traceability matrixes enables configuration management to determine the relationship between any element in design and the requirements.

10. The next lecture will present a method for annotating the algorithm of an objects methods or operations.

PROCEDURE:

teaching method and media:

vocabulary introduced:
object traceability matrix
class dictionary
Nassi Shneiderman model
Data Structure Design Trace Matrix

INSTRUCTIONAL MATERIALS:

overheads:

L31OH1	Outline
L31OH2	Definitions
L31OH3	Rumbaugh Model
L31OH4	Object oriented preliminary design
L31OH5	Object Traceability Matrix
L31OH6	Class Dictionary
L31OH7	Object oriented detailed design
L31OH8	Data structure dictionary
L31OH9	Detailed design traceability matrix
L31OH10	Detailed design deliverables
L31OH11	Preliminary design deliverables

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 025 - **Resolution of outstanding issues from last semester**

READING ASSIGNMENTS:
Mynatt Chapter 1 (pp. 31-42)
Mynatt Chapter 3 (pp. 94-138)
Mynatt Chapter 4 (pp. 169-183)

RELATED READINGS:
Schach Chapter 10 (pp. 321-324)

OUTLINE

Design

- High Level Design (Preliminary Design)
- Low Level Design (Detailed Design)

Steps in Low Level Design

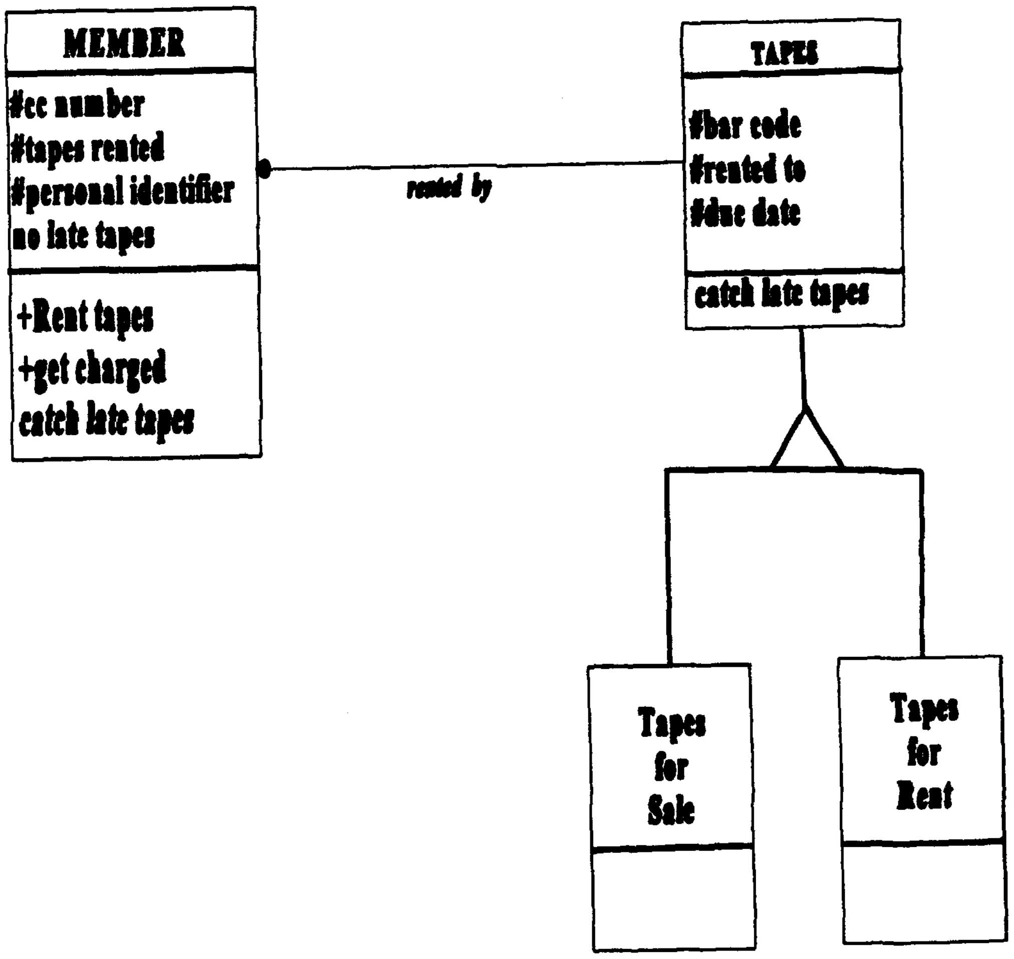
Notation to Express the Low Level Design Model

Detailed Design Deliverables

Definitions

OBJECT CLASS	Models "things" in the world: the model has attributes, operations, and a precise interface that receive messages. (a factory waiting to create instances)
INSTANCE	An actual object waiting to perform services and having state. (the object)
ENCAPSULATION	An object's state data cannot be directly accessed, it can only be asked for a service.
MESSAGE	The only way objects communicate and request services from other objects.
METHOD	An object class's service or behavior in response to a message.

Rumbaugh Model



OBJECT-ORIENTED PRELIMINARY DESIGN

High Level Design

Input	Requirements Documents
	Costumer Interviews
	Domain Analysis
Process	Identify Domain Object Classes

	Class Design
	    Divide System Responsibilities
	    Design Interfaces

	Identify Object Relationships
	Design Validation
Output	Preliminary design deliverables

Object Traceability Matrix

Functional Requirement id.	Object Name	Ada Package
1	MEMBER	MEMBERSHIP_ OPS
.		
.		
.		

CLASS DICTIONARY

OBJECT CLASS NAME:

OBJECT DESCRIPTION:

ATTRIBUTE DESCRIPTION:

-
-
-
-

Method Descriptions:

-
-
-
-

Input information needed from other objects:

-
-
-
-

OBJECT-ORIENTED DETAILED DESIGN

Low Level Design

Input	Preliminary Design Deliverables
Process	Identify Internal aspects of Objects

- Data Structure Design
  - Data Structure Dictionary
- Algorithms for Operations
  - Nassi-Shneiderman Models

Identify Object Relationships
((Update the Object Model with permission from CM))

Output	Detailed design deliverables

Data Structure Dictionary

OBJECT CLASS NAME:

Ada PACKAGE NAME:

DATA STRUCTURE NAME:

ATTRIBUTE(S) COVERED:

DESCRIPTION:

use data dictionary notation

suggest a data type

Detailed Design Traceability Matrix

DATA STRUCTURES

Data Structure Name	Objects and Attributes	Ada Package
Member_Record	MEMBER_NAME MEMBER_FEE	MEMBERSHIP _OPS
.		
.		

OPERATIONS

Nassi-Shneiderman Model Name	Object_ Operations	Ada Package
ENROLL_MEMBER	ENROLL	MEMBERSHIP_OPS
.		
.		

DETAILED DESIGN DELIVERABLES

Data Structure Design

- Data Structure Dictionary
  - object entries
  - attribute entries

Algorithm Design

- Nassi-Shneiderman (NS) models for each operation

Traceability Matrix

- Data structures are related to Object_Attribute

- NS models are related to Object_Operations

Detailed user interfaces

- Report Formats

- Low level menu content, e.g., categories of menu information

Preliminary Design Deliverables

An Object Model:

An complete object diagram using Rumbaugh notation as presented in class.

A Class Dictionary entry for each object.

An Object-Requirements traceability matrix.

Descriptions of all major user interfaces, e.g., menu formats and responses

Ada Specifications for each object class.

LECTURE NUMBER: 032

TOPIC(S) FOR LECTURE:
Reuse

INSTRUCTIONAL OBJECTIVE(S):

1. To understand the role of reuse in the development of software.
2. To learn the language features of Ada which support reuse.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

One of the primary considerations in design is reuse. This can be achieved by the design and development of reusable modular components. Reuse supports maintainability, testability, and consequently reduces the amount of effort needed to develop quality systems.

(Learning Label- Today we are going to learn ...)

In today's lecture we will examine the issue of reuse.

CONTENTS:

1. Introduction to the concept of reuse L32OH1

   a. Reuse, according to the IEEE definition, is the extent to which a module can be used in multiple applications. Usually, we are talking about the reuse of code, but reuse throughout the life cycle would also increase productivity. Reuse of design, part of a manual, or set of test data are examples of other types of reuse which may become more common in the future. In relation to code, reuse is the use of components of one product in order to facilitate the development of a different product with different or similar functionality.

   b. As will be seen later in the Ada examples, components must be generalized in order to provide for reuse. Components must also be written so that they are understandable, documented according to a set of organizational standards, and portable. L32OH2
   These attributes are needed if the components are to be understood and adaptable. Designing for reuse is more time-consuming than designing for a particular functionality; therefore, there needs to be an organizational policy decision

for reuse to be successful. Project managers must be willing to invest extra effort for long-term benefits instead of working only towards immediate results.

c. Increased productivity is a primary benefit of reuse. Increased productivity which equates to savings in time, effort, and cost during development is a result of having fewer components to design, implement, and validate. Other benefits are shown on overhead.

L32OH3

d. However, there are several problems hindering widespread reuse in industry in the United States (reuse is more common in Japan and Europe). These problems are shown on overhead L32OH4.

e. Four main types of reuse of code are application systems, subsystems, modules or objects, and functions. L32OH5 Reuse of application systems is the portability of application systems over a range of machines. This type of reuse is commonly dcne already. Reuse of functions is also already commonly accomplished; for example, a library of mathematical functions is a common feature for programming environments.

2. Ada supports reuse in these ways: generics, passing subprograms, and unconstrained arrays L32OH6.

   a. A generic unit, which is a parameterized template, provides for reuse. The instantiation of a generic tailors that generic to a specific function. Ada's support generics for procedures, functions, and packages. There are three aspects to defining and using generics: generic unit declaration, generic subprogram or package body, and instantiation of an instance of the generic unit. L32OH7, L32OH8, L32OH9, L32OH10

   L32OH7 is an example of a generic procedure to interchange two items. L32OH8-L32OH10 is an example of a generic stack package.

   b. Compilation units can be reused in different contexts based on the passing of types and subprograms as parameters. L32OH11, L32OH12, L32OH13 is an insertion sort 'which demonstrates this.

c. The use of unconstrained arrays allows arrays to be dynamically allocated; instead of having to specify the dimensions at compile time as in Pascal. L32OH14

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

<u>vocabulary introduced</u>:

reuse
generic
instantiation
unconstrained array

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>:

L32OH1	Reuse
L32OH2	Factors in implementing reuse
L32OH3	Benefits of reuse
L32OH4	Problems hindering reuse
L32OH5	Types of reuse
L32OH6	Ada and reuse
L32OH7	Example of generic procedure
L32OH8	Example of generic package specification
L32OH9	Example of generic package body
L32OH10	Example of generic instantiation
L32OH11	Example of passing types and subprograms package specification
L32OH12	Example of passing types and subprograms package body
L32OH13	Example of passing types and subprograms instantiation
L32OH14	Example of Pascal array

<u>handouts</u>:

<u>RELATED LEARNING ACTIVITIES</u>:
(labs and exercises)

Lab 026 - Reorganization of extended project

READING ASSIGNMENTS:

Sommerville Chapter 16 (pp. 309-328)
Benjamin Chapter 9 and 12 (pp. 79-85 and 111-117)

RELATED READINGS:

Berzins Chapter 8 (pp. 487-492)
Booch Chapter 14 (pp. 253-254)
Booch(2) Chapter 12 (pp. 252-255)
Ghezzi Chapter 2 (pp. 28-29)
Pressman Chapter 1 (pp. 13-15)
Schach Chapter 15 (pp. 483-484)

Reuse

The extent to which a module can be used in multiple applications [IEEE]

Practice of taking portions of previously developed software and using them in new software, perhaps with some minor alterations

Accomplished in design -- design for reuse

Factors in Implementing Reuse

Components must be generalized to satisfy a wider range of requirements

Requires an organizational policy decision to increase short-term costs for long-term gain

Components should be understandable, documented according to a set of organizational standards, and portable

Benefits of Reuse

Considerable savings in time, effort, and cost during development

Increase in system reliability

Reduction in overall risk

Effective use of specialists

Embodiment of organizational standards in reusable components

Problems Hindering Reuse

Need for properly catalogued and documented base of reusable components

Organizations are reluctant until the cost-effectiveness of reuse is demonstrated

CASE tools do not support reuse

Confidence level among software engineers for reusable components is still low

Legal issues over ownership of contract software

Types of Reuse

Application systems

Subsystems

Modules or objects

Functions

Ada and Reuse

Generics

Passing of types and subprograms as parameters

Unconstrained arrays

Example of Generic Procedure

```
--header
generic
   type ITEM is private;
procedure INTERCHANGE (FIRST, SECOND :
                          in out ITEM);

----------------------------------------

--body
procedure INTERCHANGE (FIRST, SECOND :
                          in out ITEM) is
   TEMP : ITEM;
begin
   TEMP := FIRST;
   FIRST := SECOND;
   SECOND := TEMP;
end;

----------------------------------------

--instance
procedure INTEGER_INTERCHANGE is new
   INTERCHANGE (ITEM => integer);

--instance

procedure BOOLEAN_INTERCHANGE is new
   INTERCHANGE (ITEM => boolean);
```

Example of Generic Package Specification

```
generic
   SIZE : positive;
   type ITEM is private;
package STACK is
   procedure PUSH (X : in ITEM);
   procedure POP (X : out ITEM);

   STACK_OVERFLOW,
   STACK_UNDERFLOW : exception;
end STACK;
```

Example of Generic Package Body

```
package body STACK is
   SPACE : array (1..SIZE) of ITEM;
   INDEX : integer range 0..SIZE := 0;

   procedure PUSH (X : in ITEM) is
   begin
      if INDEX = SIZE then
         raise STACK_OVERFLOW;
      else
         INDEX := INDEX + 1;
         SPACE (INDEX) := X;
      end if;
   end PUSH;

   procedure POP (X : out ITEM) is
   begin
      if INDEX = 0 then
         raise STACK_UNDERFLOW;
      else
         X := SPACE (INDEX);
         INDEX := INDEX - 1;
      end if;
   end POP;
end STACK;
```

Example of Generic Instantiation

```
package INTEGER_STACK is
   new STACK (10, integer);

-----------------------------------------------------

type EMPLOYEE is
   record
      NAME : string (1..40);
      ID   : integer;
   end record;

package EMPLOYEE_STACK is
   new STACK (SIZE => 25,
              ITEM => EMPLOYEE);
```

Example of Passing Types and Subprograms Procedure Specification

```
generic
  type ITEM is private;
   type VECTOR is array (integer range <>) of
ITEM;
  with function ">" (A,B : ITEM)
     return BOOLEAN is <>;
procedure  INSERTION_SORT(A  :  in  out
VECTOR);
```

Example of Passing Types and Subprograms
Procedure Body

```
procedure INSERTION_SORT(A : in out VECTOR) is
  I, J : integer;
  T    : ITEM;
  L    : integer := A'first;
  U    : integer := A'last;
begin
  I := L;
  while I /= U loop
    T := A (I + 1);
    J := I + 1;
    while J /= L and then A (J - 1) > T loop
      A (J) := A (J - 1);
      J := J - 1;
    end loop;
    A (J) := T;
    I := I + 1;
  end loop;
end INSERTION_SORT;
```

Example of Passing Types and Subprograms Instantiation

```
type EMPLOYEE is
  record
    NAME : string (1..40);
    ID   : integer;
  end record;

type EMPLOYEE_ARRAY is array (integer
                                  range <>)
   of EMPLOYEE;

function ">" (A,B : EMPLOYEE) return boolean is
begin
   return A.ID > B.ID;
end;

procedure EMPLOYEE_SORT is new
  INSERTION_SORT (ITEM => EMPLOYEE,
                  VECTOR => EMPLOYEE_ARRAY,
                  ">" => ">");
------------------------------------------------------
procedure EMPLOYEE_SORT is new
  INSERTION_SORT (EMPLOYEE,
                  EMPLOYEE_ARRAY, ">");
```

Example of Pascal Array

```
const
  SIZE = 17;
type
  MATRIX = array [1..SIZE, 1..SIZE] of char;

procedure TRANSPOSE_MATRIX
  (IN_MATRIX      : MATRIX;
      var OUT_MATRIX : MATRIX);
```

Example of Ada Unconstrained Array

```
type MATRIX is array (integer range <>,
                      integer range <>) of character;

procedure TRANSPOSE_MATRIX
  (IN_MATRIX  : in  MATRIX;
     OUT_MATRIX : out MATRIX);
```

LECTURE NUMBER: 033

TOPIC(S) FOR LECTURE:
Nassi-Shneiderman Chart notation

INSTRUCTIONAL OBJECTIVE(S):

1. To learn Nassi-Shneiderman Chart notation for representing algorithms used in detailed design.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

We have begun looking at detailed design. The algorithms developed in detailed design must be clearly communicated at a high level of abstraction but which can also be easily implemented. One such notation is Nassi Shneiderman diagrams.

(Learning Label- Today we are going to learn ...)

Today we will look at a notation for representing detailed design.

CONTENTS:

1. There are a number of ways to represent algorithms. Although the most commonly used is pseudo-code, we chose to use Nassi Shneiderman diagrams. We chose them in order to discourage students from writing code during design. Discuss the advantages and disadvantages of Nassi Shneiderman charts (NSC). L33OH1

L33OH2

2. Present the students with the notation and rules for creating NSC.

3. The three basic constructs for representing any algorithm are sequence, selection, and iteration. Discuss the Nassi-Shneiderman Chart notation for these are shown in overhead L33OH3.

4. Discuss the more complicated examples given in overheads L33OH4, L33OH5, L33OH6, L33OH7.

PROCEDURE:
teaching method and media:

Lecture and overheads are the chief media for this lecture.

vocabulary introduced:

INSTRUCTIONAL MATERIALS:

overheads:

L33OH1	Nassi-Shneiderman Charts
L33OH2	Rules for drawing and creating Nassi-Shneiderman charts
L33OH3	Notation for sequence, selection, and iteration
L33OH4	Example notation for complex IF statements
L33OH5	More Complex IF statements
L33OH6	Example notation for CASE statements
L33OH7	Example notation for a procedure
L33OH8	Example notation for DO WHILE loop

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 027 - Nassi-Shneiderman charts
Preparation for detailed design review

Use Lab to have the students practice NSCs.

READING ASSIGNMENTS:
Mynatt Chapter 5 (pp. 198-202)

RELATED READINGS:
Pressman Chapter 10 (pp. 345-350)

Nassi-Shneiderman Charts

Method for representing structured algorithms

Advantages:

Easy to learn

Easy to read
Easy to convert to source code

Good at encouraging structured design
Flexible in the level of detail shown

Standardized

Disadvantages:

Difficulty in drawing and modifying if no access to CASE tool that provides this notation

Rules for Drawing and Creating Nassi-Shneiderman Charts

The charts are always rectangular.

The flow of control in a Nassi-Shneiderman chart always starts at the top.

Flow of control always moves from top to bottom, except when iteration is involved.

Vertical lines may never be crossed.

A rectangle may be exited in a downward direction only.

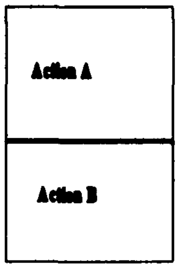
A rectangle may be empty, representing null or empty action.

A rectangle may represent another Nassi-Shneiderman chart (for example, a call to another procedure).

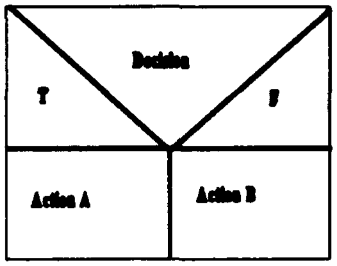
(Mynatt)

Nassi-Shneiderman Notation

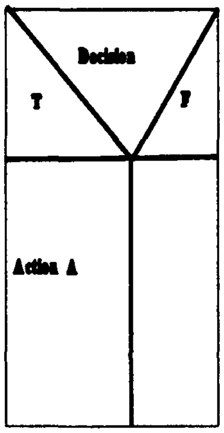
a. Sequence



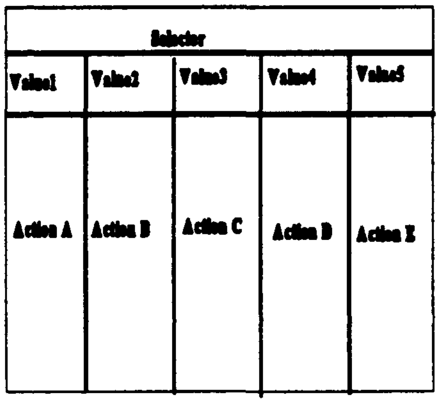
b. Decision (if-then-else)



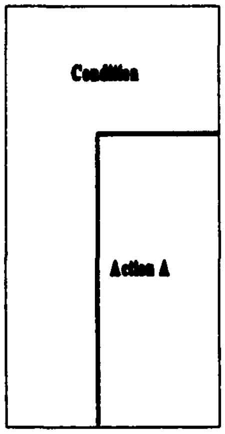
c. Decision (if-then)



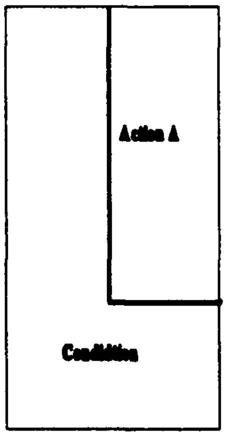
d. Selection (case)



e. Iteration (While)

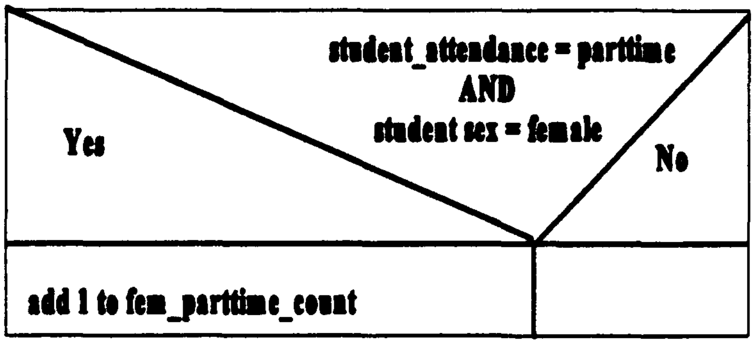


f. Iteration (repeat- until)



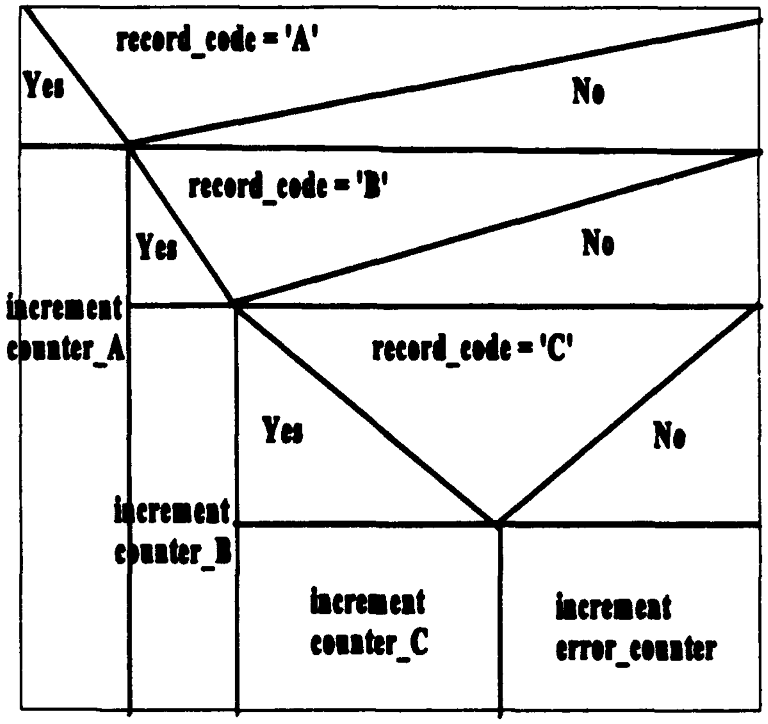
Nassi-Shneiderman Notation For Complex If Statements

IF statement (multiple conditions)



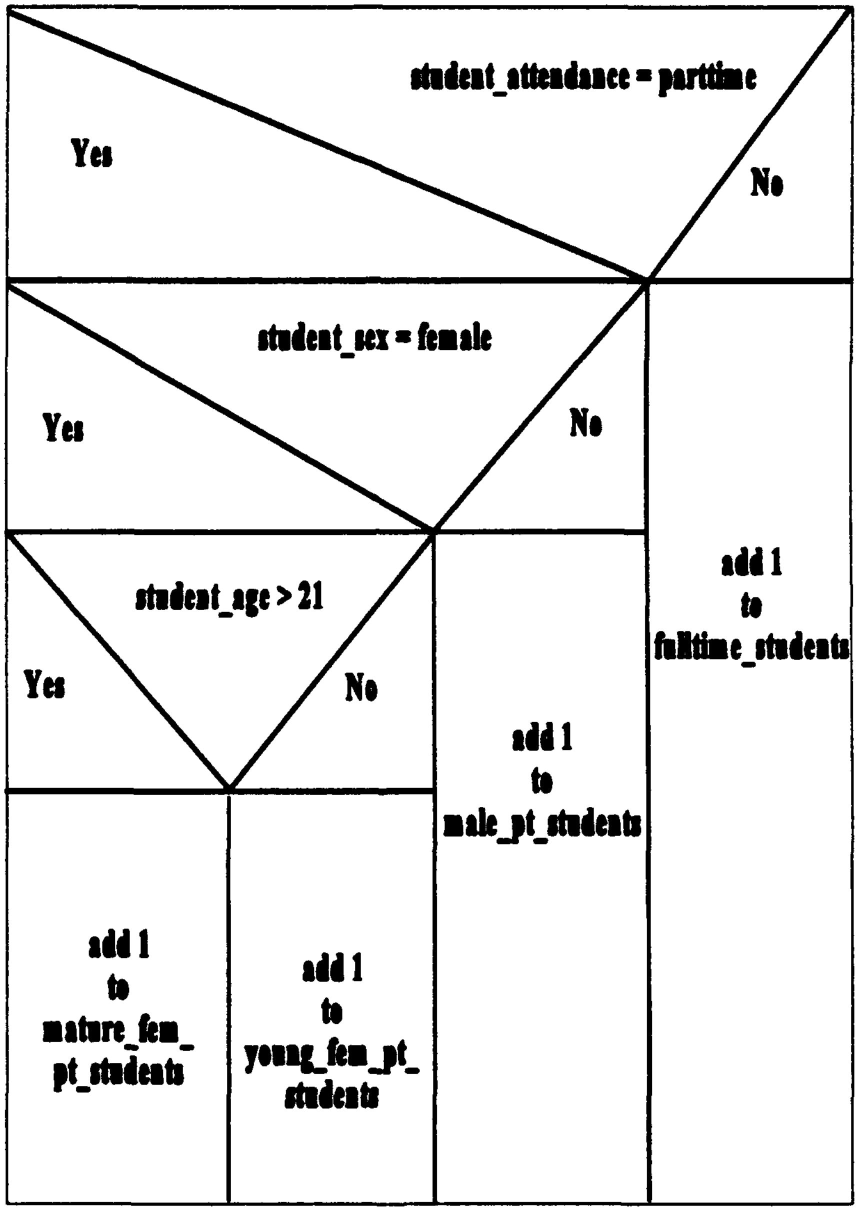
NESTED IF STATEMENT

a. Linear nested IF statement

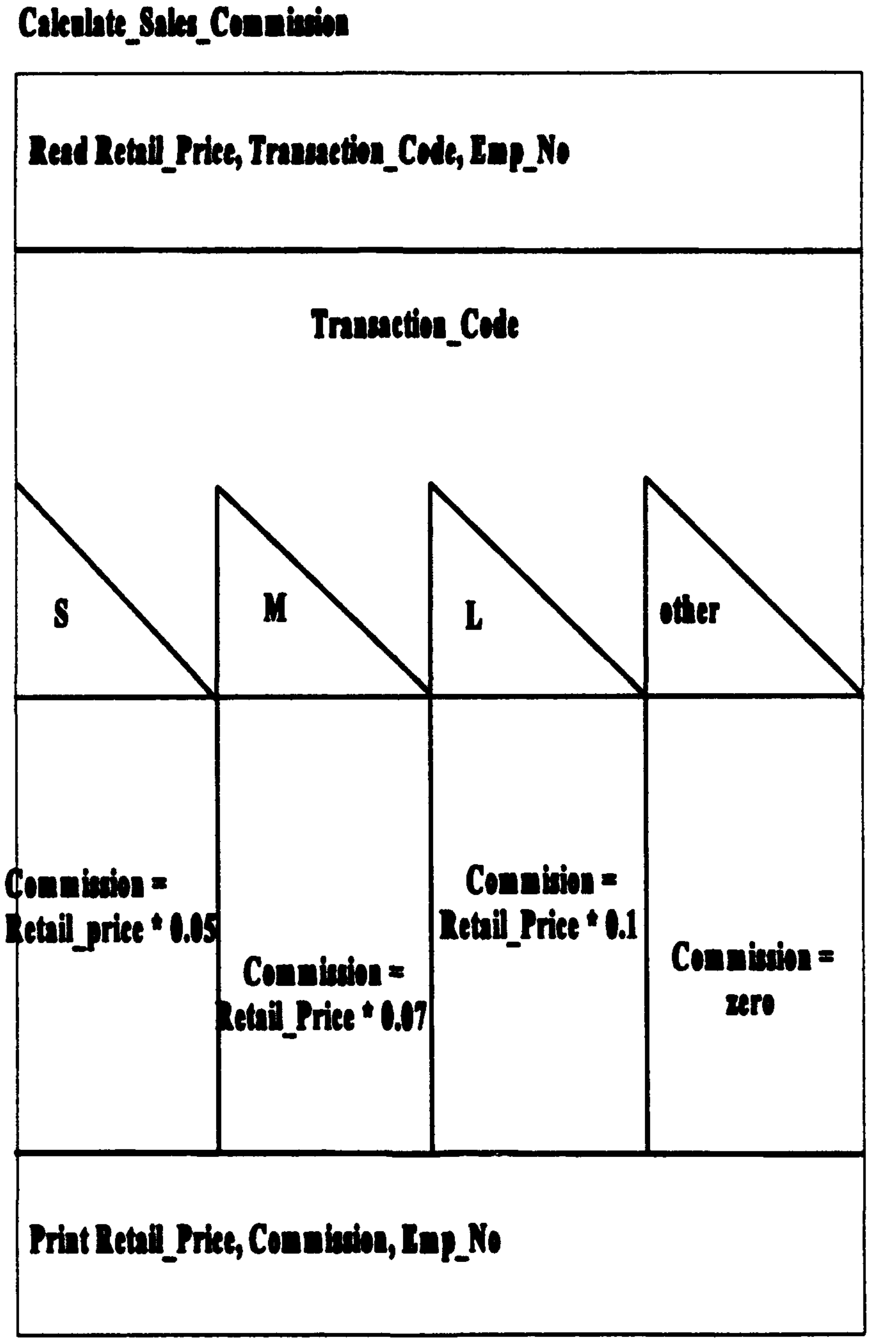


Nassi-Shneiderman Notation
Another Example of a Complex IF Statement

b. Non-linear nested IF statement

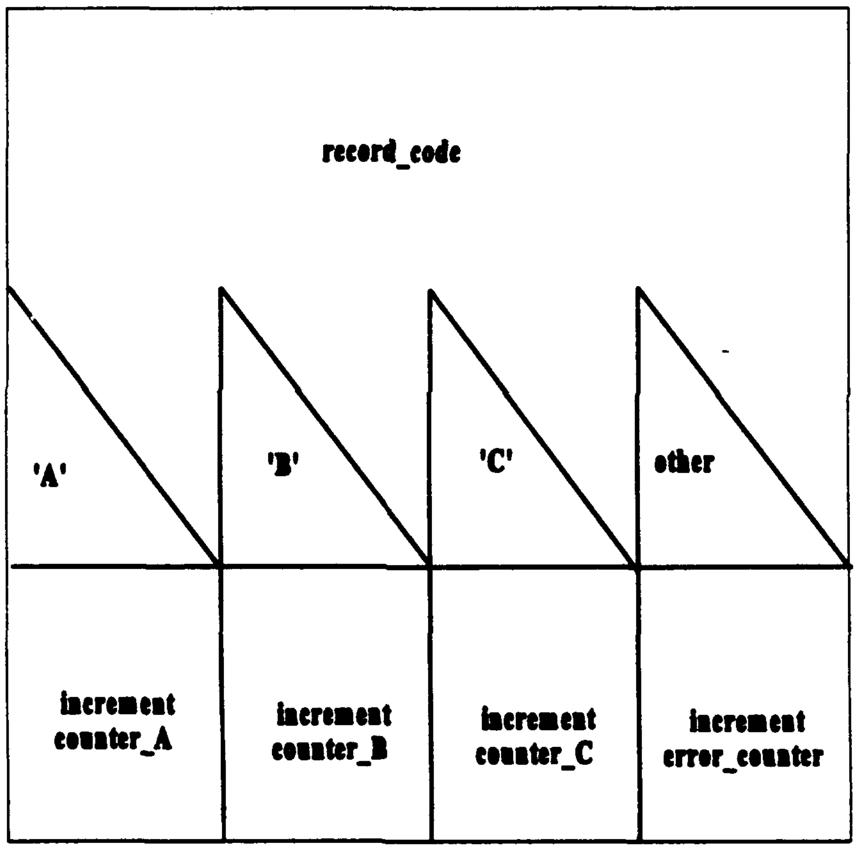

student_attendance = parttime
Yes
No
student_sex = female
Yes
No
student_age > 21
Yes
No
add 1 to mature_fem_pt_students
add 1 to young_fem_pt_students
add 1 to male_pt_students
add 1 to fulltime_students

Nassi-Shneiderman Notation for CASE Statements



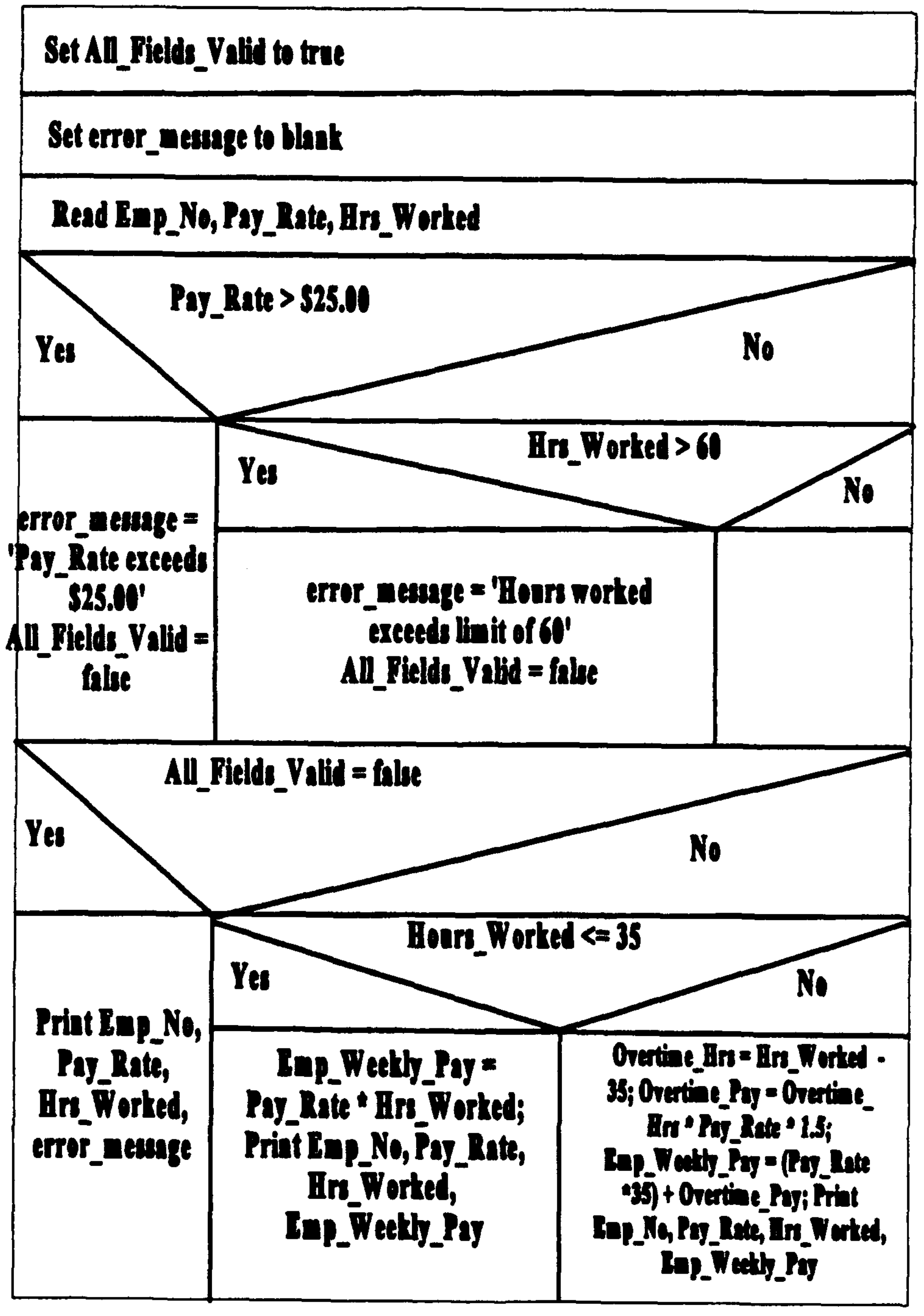
Calculate_Sales_Commission
Read Retail_Price, Transaction_Code, Emp_No
Transaction_Code
S
M
L
other
Commission = Retail_price * 0.05
Commission = Retail_Price * 0.07
Commision = Retail_Price * 0.1
Commission = zero
Print Retail_Price, Commission, Emp_No

Nassi-Shneiderman Notation for Case Statements

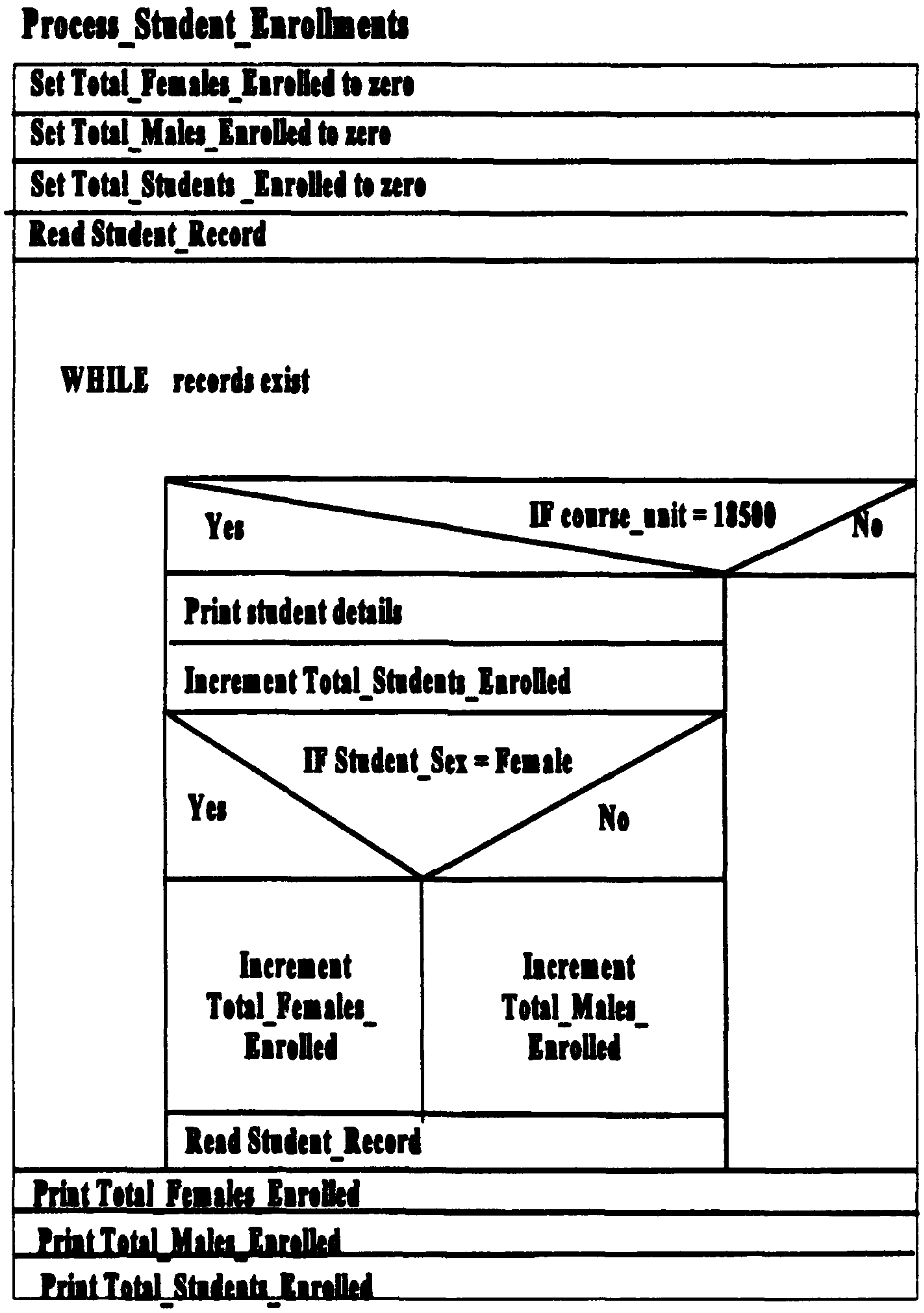


Nassi-Shneiderman Notation for Nested IF Statements

Compute_Employee_Pay


Set All_Fields_Valid to true
Set error_message to blank
Read Emp_No, Pay_Rate, Hrs_Worked
Pay_Rate > $25.00
Yes
No
error_message = 'Pay_Rate exceeds $25.00' All_Fields_Valid = false
Hrs_Worked > 60
Yes
No
error_message = 'Hours worked exceeds limit of 60' All_Fields_Valid = false
All_Fields_Valid = false
Yes
No
Print Emp_No, Pay_Rate, Hrs_Worked, error_message
Hours_Worked <= 35
Yes
No
Emp_Weekly_Pay = Pay_Rate * Hrs_Worked; Print Emp_No, Pay_Rate, Hrs_Worked, Emp_Weekly_Pay
Overtime_Hrs = Hrs_Worked - 35; Overtime_Pay = Overtime_Hrs * Pay_Rate * 1.5; Emp_Weekly_Pay = (Pay_Rate *35) + Overtime_Pay; Print Emp_No, Pay_Rate, Hrs_Worked, Emp_Weekly_Pay

Nassi-Shneiderman Notation For DOWHILE Loops


Process_Student_Enrollments
Set Total_Females_Enrolled to zero
Set Total_Males_Enrolled to zero
Set Total_Students _Enrolled to zero
Read Student_Record
WHILE records exist
IF course_unit = 18500
Yes
No
Print student details
Increment Total_Students_Enrolled
IF Student_Sex = Female
Yes
No
Increment Total_Females_ Enrolled
Increment Total_Males_ Enrolled
Read Student_Record
Print Total Females Enrolled
Print Total_Males_Enrolled
Print Total_Students_Enrolled

LECTURE NUMBER: 034

TOPIC(S) FOR LECTURE:
Introduction to Ada and I/O in Ada

INSTRUCTIONAL OBJECTIVE(S):

1. To understand the uniqueness of Ada.
2. To learn the elementary features of the Ada programming language and how it handles input and output.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

You will begin the implementation phase of the second project soon. This project is to be implemented in Ada. In previous classes, we have looked at Ada software artifacts through program reading.

(Learning Label- Today we are going to learn ...)

Now we will look in detail at Ada in order to prepare for the implementation phase of the second project.

CONTENTS:

L34OH1

1. Discuss the general characteristics of Ada. Although it was built for embedded systems , it is a general purpose language.

2. It is assumed that students are already proficient in a standard strongly typed programming language. The implementation details of Ada will be covered quickly to enable students to start work on their project implementation. This rapid introduction is done by frequent references/comparisons to the language in which they are already proficient.

3. Standard input and output with Text_IO L34OH2

   a. Ada contains no input and output standards; instead, Ada provides a general-purpose package called Text_IO which contains a collection of subprograms, types, subtypes, and exceptions used for file manipulation and input and output to keyboard and monitor. The Text_IO package provides directly the routines needed for input and output of character and string data types. Included in Text_IO are generic packages for input and output of integers, floating point numbers, fixed point numbers, and enumeration data type. There are Ada I/O packages for manipulating other data types as well.

b. L34OH3 shows the basic routines provided by Ada along with their equivalent Pascal statement.

   i Get is analogous to Pascal's Read. Get_Line however reads strings only and requires a second parameter into which the length of the line read is placed.

   ii Similarly, Put_Line is available only for the string data type. Put needs formatting information for integers and decimal numbers. Point this out in the examples shown. X (in fourth example) is an integer and Y (in fifth example) is of type float. Point out also that Ada enables you to specify the base of the number being printed.

c. L34OH4
Discuss the sample program shown.

   i. Note the following: "with" and "use" are parts of the "context clause"; The "use" clause cause the compile to look at Text_IO for commands; so that one need only write PUT("Hello world!"); rather than Text_IO.PUT("Hello world");

   ii. The "with" clause tells the compiler that the procedure needs the help of another package, and specifies that package name.

d. Text_IO also provides standard input and output for text files. L34OH5 A text file is a collection of characters that are organized into lines and pages and is terminated with a file terminator. The data type for a text file, as provided in Text_IO, is File_Type. There are two file modes for text files: In_file for reading data from a file and Out_File for writing data to a file. Every text file must be specified in one of these two modes.

e. Text_IO provides several routines for the management of files including Create, Open, Close, Delete, Reset, End_of_Line, and End_of_File. Discuss these using overheads L34OH6, L34OH7, L34OH8.

f. Exceptions are abnormal conditions that occur at run time. Discuss some of the commonly used exceptions shown on overheads L34OH9, L34OH10.

g. Discuss the sample program illustrating file usage is shown on overhead L34OH11.

PROCEDURE:

teaching method and media:

Lecture and overheads are the chief media for this lecture.

vocabulary introduced:

Text_IO
text file
file mode

INSTRUCTIONAL MATERIALS:

overheads:

L34OH1	Ada
L34OH2	Standard input and output with Text_IO
L34OH3	Text_IO Operations
L34OH4	Sample program using Text_IO
L34OH5	Standard input and output on files
L34OH6	File management
L34OH7	File management(cont.)
L34OH8	File management(cont.)
L34OH9	Exceptions provided by Text_IO
L34OH10	Exceptions provided by Text_IO(cont.)

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

READING ASSIGNMENTS:

Benjamin Chapters 1 (pp. 1-10)

RELATED READINGS:

Booch Chapter 6 (pp. 69-80)
Booch(2) Chapter 4 (pp. 60-71)

Ada

General-purpose

Strongly typed

Block-structured

Procedural

Supports concurrency, exception handling, and low-level, implementation-dependent features

Designed to support software engineering concepts
- Information hiding
- Modularity
- Reusability
- Maintainability

Standard Input and Output with Text_IO

Text_IO is package of routines provi ınput and output on text files, keyboard, and monitor

Directly provides routines for input and output of character and string types

Provides generic packages for:
- Integer (**Integer_IO**)
- Float (**Float_IO**)
- Fixed-point (**Fixed_IO**)
- Enumeration (**Enumeration_IO**)

Text_IO Operations

Ada Routine	Pascal Equivalent
Get (A);	read (A);
Get_Line (A, Length);	readln (A);
Put ("hello");	write ('hello');
Put (X,2,10);	write (x:2);
Put (Y,3,2,0);	write (y:6:2);
Put_Line ("hello");	writeln ('hello');
Skip_Line;	readln;
Skip_Line (2);	readln; readln;
New_Line;	writeln;
New_Line (2);	writeln; writeln;

Sample Program using Text_IO

```
with Text_IO;  use Text_IO;
procedure POWER is
   package INT_IO is new Integer_IO (integer);
   use INT_IO;
     BASE, COUNT, EXPONENT, PRODUCT   :
integer;
begin
   Put_Line ("Please enter the base value");
   Get (BASE);
   Put_Line ("Please enter the exponent value");
   Get (EXPONENT);
   COUNT := 1;
   PRODUCT := BASE;
   while COUNT < EXPONENT loop
      PRODUCT := PRODUCT * BASE;
      COUNT := COUNT + 1;
   end loop;
   Put ("Base = ");
   Put (BASE);
   Put ("   Exponent = ");
   Put (EXPONENT);
   Put ("Product = ");
   New_Line;
   Put (PRODUCT);
end POWER;
```

Standard Input and Output on Files

Text file

A collection of characters that may be organized into lines and pages (using line terminators and page terminators). The end of a text file is indicated with a file terminator.

Supported in **Text_IO**

Data type called **File_Type**

File modes available for text files:

In_File indicates a file is read only

Out_File indicates a file is write only

File Management

Create

Opens a new external file and associates an internal file with it; file is initially empty

Default file mode is **Out_File**

Parameters:	1 - file identifier
	2 - access mode
	3 - external file name

Example of use:

```
Create (Report_file, Out_File,
        "a:\summary.rep");
```

Open

Opens an existing file for processing, starting at the beginning of the file

No default file mode

If the external file specified does not exist, a **Name_Error** exception is raised

```
Open (Source2, In_File,
      "a:\prog1jones.pas");
```

File Management (cont.)

Close

Removes the association of the Ada file identifier with its associated external file

Close (Source);

Delete

Deletes the external file associated with the given Ada file identifier

Delete (Report_file);

Reset

Moves back to the beginning of the file, possibly changing the file mode, and allowing reading or writing operations to resume from the beginning of the file. The file mode is changed only if a new mode is specified as the second parameter.

Reset (Source1);

Reset (Source2, Out_File);

File Management (cont.)

Boolean functions to test for the current position in reading input:

End_of_Line	Returns true if the next component is a line terminator or file terminator; otherwise, returns false **End_of_Line (Source1);**
End_of_Page	Returns true if the next component is a page terminator or file terminator; otherwise, returns false **End_of_Page (Source1);**
End_of_File	Returns true if the next component is either a file terminator or the three-component sequence of line terminator, page terminator, file terminator; otherwise, returns false **End_of_File (Source1);**

Exceptions Provided by Text_IO

Status_Error	Attempt to use a file that has not been opened or to open a file that is already open
Mode_Error	Attempt to perform an input operation on a file of mode **Out_File** or to perform an output operation on a file of mode **In_File**
Name_Error	Attempt to associate an internal file with an external file if an invalid external file name is specified
Use_Error	Attempt to perform some input/output operation on an external file for which implementation does not allow that operation

Exceptions Provided by Text_IO (cont.)

Device_Error	A problem with the hardware, software, or media providing input/output services
End_Error	Attempt to read past end of an input file
Data_Error	Input data that is not of expected form
Layout_Error	Invalid **Text_IO** formatting operations

Sample Program using Files

```
with Text_IO;
use Text_IO;

procedure MAIN is

   package INT_IO is new Integer_IO (integer);
   use INT_IO;

   INPUT_FILE  : File_Type;
   NEXT_ITEM   : integer;

begin
   open (INPUT_FILE, In_File, "a:\testresults.dat");

   while not End_of_File (INPUT_FILE) loop
      Get (INPUT_FILE, NEXT_ITEM);
      Put ("Test result = ");
      Put (NEXT_ITEM);
      New_Line;
   end loop;

   Close (INPUT_FILE);
end MAIN;
```

LECTURE NUMBER: 035

TOPIC(S) FOR LECTURE:
Data types in Ada

INSTRUCTIONAL OBJECTIVE(S):

1. To introduce the concept of a compilation unit.

1. To learn the features of the Ada programming language concerning scalar data types.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)
(Learning Label- Today we are going to learn ...)

CONTENTS:

1. Ada's program structure L35OH1

   a. Comments in Ada by begin with a double hyphen "--" . Comments terminate at the end of a line. There is no other comment terminator.

   b. Compilation units are the pieces of a program that can be compiled separately. Compilation units include package specifications, package bodies, subprogram declarations, and subprogram bodies. The Ada compiler maintains a program library with the needed information about the compilation units used in a program. These compilation units can be specified in the context clause. Ada can use this information to provide consistency checking across the separate compilation units. The main program does not have any parameters.

2. Ada data types L35OH2

   a. Ada provides scalar data types (discrete and real), composite data types (arrays and records), access data types, private data types, subtypes and derived types. Today we are only examining the scalar data types of Ada. The discrete data types to be examined are represented internally as integer (Ada provides predefined integer types **Integer**, **Natural**, **Positive**) and enumeration (Ada provides predefined enumeration types **Character** and **Boolean**). The real data types provided by Ada are floating point (**Float**) and fixed point. For each of these data types, we will look at the predefined range of values for Ada's predefined data types, user-defined data types, declaration of objects, declaration of constants, operations and operators, attributes, and input/output. Universal integers and universal reals are also discussed. Review the overheads. L35OH12 is an exercise to use in class. There will be constraint errors on Pred(Monday), Succ(Sunday) and Val(7). L35OH3, L35OH4, L35OH5, L35OH6, L35OH7, L35OH8, L35OH9, L35OH10, L35OH11, L35OH12, L35OH13, L35OH14

PROCEDURE:

teaching method and media:

vocabulary introduced:

INSTRUCTIONAL MATERIALS:

overheads:

L35OH1	Ada's program structure
L35OH2	Ada's data types
L35OH3	Integer data type
L35OH4	Named numbers
L35OH5	Real numbers
L35OH6	Formatting numeric output
L35OH7	Arithmetic operations
L35OH8	Numeric attributes
L35OH9	Enumeration data type
L35OH10	Enumeration I/O
L35OH11	Attributes for enumeration data type
L35OH12	Examples of enumeration attributes
L35OH13	Derived type declarations
L35OH14	Subtype declarations

handouts:

__RELATED LEARNING ACTIVITIES__:
(labs and exercises)

__READING ASSIGNMENTS__:
Benjamin Chapters 2-3 (pp. 11-28)

__RELATED READINGS__:

Booch	Chapter 8 (pp. 103-115)
Booch(2)	Chapter 6 (pp. 93 - 105)

Ada's Program Structure

Comments

Begins with a double hyphen "--" and extends to the end of the line

Compilation units

Pieces of a program that can be compiled separately

May be package specification, package body, subprogram declaration, or subprogram body

Ada keeps a library ("program library") with information about the compilation units used in a program; thus, Ada can check for consistency between the compilation units

Context clause

Main programs

Only parameterless procedures

Ada's Data Types

Specify a set of values and a set of operations applicable to these values

Provided by Ada:
- Scalar
  - Discrete
    - Integer (**Integer**, **Natural**, **Positive**)
    - Enumeration
      - Character (**Character**)
      - Boolean (**Boolean**)
  - Real
    - Floating point (**Float**)
    - Fixed point
- Composite
  - Arrays
    - Constrained
    - Unconstrained
    - Strings (**String**)
  - Records
- Access (i.e., pointers)
- Subtype and derived types

Integer Data Type

Predefined range of values

Integer'first..Integer'last

Subtypes of integers

Predefined subtypes

```
Natural     integer >= 0
subtype NATURAL is integer
    range 0..integer'last;

Positive    integer >= 1
subtype Positive is integer
    range 1..integer'last;
```

User-defined subtypes

```
type INDEX is range 1..50;
```

Declaring integers

```
NUM       : Integer;
INCREMENT : Integer := 1;

-- constant must be initialized
DECREMENT : constant Integer := 1;

A,B,C     : integer := 0;
X,Y,Z     : integer := 2,3,4; -- illegal
```

Named Numbers

A constant that is declared without assuming a type and can be used with all numeric types.

```
TENS      : constant := 10;
HUNDREDS : constant := TENS * 10;
```

Real Numbers

Floating Point

Relative error

Predefined range of values

Float'first..Float'last

User-defined subtypes

```
type CELSIUS is digits 3;
type DISTANCE is digits 3 range -50..50;
```

Declaring floats

```
ROOM_RATE   : Float;
PI          : constant Float := 3.1456;
```

Fixed Point

Must define accuracy specification (absolute error) and range in declaration

```
type RATE is delta 0.001 range 7.0..12.0;
```

Formatting Numeric Output

For integers:

```
Put (INTEGER_VALUE, width_of_field);

Put (X, 3);  -- outputs __3
        Fld               rt justified
        width
```

For reals:

Put (REAL_VALUE, fore, aft, exp);
fore=number of digits before decimal point
aft=number of digits after decimal point
exp=the base, e.g. 0=base 10

```
Put (2.573, 3, 2, 0);  -- output __2.57
```

Arithmetic Operations

Unary Arithmetic Operations

Operation	Operator
absolute value	**abs**
unary plus	+
unary minus	-

Binary Arithmetic Operations

Operation	Operator
exponentiation	**
multiplication	*
division	/
modulus	**mod**
remainder	**rem**
addition	+
subtraction	-

Relational Operators

Operation	Operator
equal	=
not equal	/=
less than	<
less than or equal	<=
greater than	>
greater than or equal	>=

Numeric Attributes

Attribute

gives information about particular properties of a type

Attributes for integers:

First	first value in integer's range
Last	last value in integer's range

put(integer'first); prints the first integer

Attributes for floats:

First	first value in float's range
Last	last value in float's range
Digits	number of significant figures
Small	smallest positive float number
Large	largest positive float number

Enumeration Data Type

Subtypes of enumeration data types
    Predefined subtypes

Boolean	false, true
Character	ASCII character set

    User-defined subtypes

```
type DAY is (MONDAY, TUESDAY,
    WEDNESDAY, THURSDAY, FRIDAY,
    SATURDAY, SUNDAY);
```

Declaring enumeration data types

```
TODAY     : DAY;
TOMORROW  : DAY := TUESDAY;
FIRST_DAY : constant DAY := MONDAY;
```

Operators
    Relational (**=, /=, <, <=, >, >=**)
    Membership (**in, not in**)
    Boolean operators (**and, or, xor, not**)

Enumeration I/O

Remember that **Text_IO** provides a generic package for enumeration input and output

```
package DAY_IO is new Enumeration_IO
(DAY);
use DAY_IO;
```

```
package BOOLEAN_IO is
    new ENUMERATION_IO (Boolean);
use BOOLEAN_IO;
```

Attributes for Enumeration Data Type

First	first value in enumeration data type
Last	last value in enumeration data type
Pred	predecessor of argument constraint error on first
Succ	successor of argument constraint error on last
Pos	position in list (count starts at 0)
Val	value associated with argument which is the position in the list

Examples of Enumeration Attributes

```
type DAY is
(MONDAY,TUESDAY,WEDNESDAY,
THURSDAY, FRIDAY, SATURDAY, SUNDAY);
```

```
DAY'First

DAY'Last

DAY'Pred (TUESDAY)

DAY'Pred (MONDAY)

DAY'Succ (TUESDAY)

DAY'Succ (SUNDAY)

DAY'Pos (TUESDAY)

DAY'Val (2)

DAY'Val (7)
```

Derived Type Declarations

defines a new and distinct type which inherits all the features of the parent type

new type is not compatible with parent type

```
with Text_IO; use Text_IO;
procedure DERIVED_DEMO is
   type LENGTH is new Integer;
   type AREA is new Integer;
   package AREA_IO is new Integer_IO
(AREA);
   use AREA_IO;
   L1, L2 : LENGTH := 3;
   A      : AREA;
   function "*" (X, Y : LENGTH) return AREA
is
   begin
      return AREA (Integer (X) * Integer (Y));
   end;

   begin
      A := L1 * L2;
      -- The "*' will only work if L1 and L2
      -- match the data type
      Put (A);
   end DERIVED_DEMO;
```

Subtype Declarations

Does not define a new (i.e., distinct) type but promotes readability

Provides a new name for another (potentially constrained) data type

Inherits all the properties of base type

```
type DAY is
(MONDAY, TUESDAY, WEDNESDAY,
THURSDAY, FRIDAY, SATURDAY, SUNDAY);

subtype WEEKDAY is DAY
    range MONDAY..FRIDAY;
```

LECTURE NUMBER: 036

TOPIC(S) FOR LECTURE:
Statements in Ada

INSTRUCTIONAL OBJECTIVE(S):

1. To learn the features of several Ada statements.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)
(Learning Label- Today we are going to learn ...)

CONTENTS:

1. Ada statements L36OH1

   a. A statement is a program construct defining an action to be performed during execution. There are two types of statements in Ada: simple (e.g., assignment statement) and compound (e.g., loop, if, case). A compound statement is a control structure that surrounds other statements whereas a simple statement does not. Every statement is terminated by a semicolon; in fact, the semicolon is considered part of the statement and not a statement separator.

   L36OH2
   b. The Ada statements examined are assignment statement, null statement, block statement, iteration statements (basic loop, for loop, while loop), and selection statements (if, case) . Discuss examples of each of these. Discuss only the unique features of Ada are discussed.

   c. The null statement uses the reserved word **null** to show a statement that performs no action. This type of statement is commonly used in case statements and exception handlers.

   d. The block statement encapsulates a collection of declarations and statements that are logically related. The scope of the declarations and exceptions within a block is the block itself. The block statement is commonly used for several purposes. It ensures that the declaration(s) within the block are not used inappropriately by other parts of the program. It documents the extent of the declaration(s), and it provides an exception handler environment. The declaration part and exception handler part are optional.

e. The simplest loop in Ada provides for an infinite loop (i.e., one that loops forever) L36OH3. This type of loop is useful for activities such as data-sampling that must continue forever once initiated. The loop can be constructed to end by including one or more **exit** statements within the loop. An unconditional or conditional **exit** statement may be used. L36OH4, L36OH5, L36OH6, L36OH7

f. Discuss the remaining Ada statements, L36OH4-L36OH7

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

Lecture and overheads are the chief media for this lecture.

<u>vocabulary introduced</u>:

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>:

L36OH1	Ada statements
L36OH2	Ada statements
L36OH3	Iteration statements - basic loop
L36OH4	Iteration statements - for loop
L36OH5	Iteration statements - while loop
L36OH6	Selection statements - if statement
L36OH7	Selection statements - case statement

<u>handouts</u>:

<u>RELATED LEARNING ACTIVITIES</u>:
(labs and exercises)

<u>READING ASSIGNMENTS</u>:

Benjamin Chapters 4 (pp. 29-37)

<u>RELATED READINGS</u>:

Booch Chapter 11 (pp. 187-197)
Booch(2) Chapter 9 (pp. 181-190)

Ada Statements

A program construct defining an action to be performed during execution

Terminated by semicolon

Two types
- Simple
- Compound

Ada Statements

Assignment statement

Null statement

Statement that performs no action

```
when others => null;
```

Block statement

Encapsulates a collection of declarations and statements that are logically related

Scope of identifiers and exceptions for block is the region of program text that begins with the declaration and extends to the end of the block

```
declare
   TEMP : Integer := NUM_1;
begin
   NUM_1 := NUM_2;
   NUM_2 := TEMP;

  -- exception handlers can go here
end;
```

Iteration Statements
Basic Loop

Structured to loop forever (one entry, no exit)

```
loop
   -- statements
end loop;
```

To leave such a loop, use the **exit** statement

```
loop
   -- statements
   exit;
   -- statements
end loop;
```

```
COUNT := 1;
loop
   COUNT := COUNT + 1;
   -- statements
   exit when (COUNT = 10);
end loop;
```

Iteration Statements
For Loop

(X is implicitly declared with the 'for')

```
for X in 1..3 loop
   -- statements
end loop;
```

```
for INDEX in 1..USER_INPUT_LIMIT loop
   -- statements
   NAME := AN_ARRAY (INDEX);
   -- statements
end loop;
```

```
for CH in 'A'..'Z' loop
   put (CH);
end loop;
```

```
for CH in reverse 'A'..'Z' loop
   put (CH);
end loop;
```

Iteration Statements
While Loop

Sequence of statements is repeated as long as condition in while condition is true

```
while not end_of_file (SOURCE2) loop
   get (SOURCE2, A);
   -- statements
end loop;
```

```
COUNT := 1;
while (COUNT /= 10) loop
   COUNT := COUNT + 1;
   -- statements
end loop;
```

```
SUM := 0;
Get (A_VALUE);
while (A_VALUE /= 0) loop
   SUM := SUM + A_VALUE;
   Get (A_VALUE);
end loop;
```

Selection Statements
If Statement

```
if VAL_1 > VAL_2 then
  MAX := VAL_1;
  Put_line ("First value is largest");
end if;

if BALANCE <= 0.0 then
  SERVICE_CHARGE := 10.00;
elsif BALANCE < 300.00 then
  SERVICE_CHARGE := 3.00;
else
  SERVICE_CHARGE := 1.00;
end if;

if BALANCE > 0.0 then
  if BALANCE < 300.00 then
    SERVICE_CHARGE := 3.00;
  else
    SERVICE_CHARGE := 1.00;
  end if;
else
  SERVICE_CHARGE := 10.00;
end if;
```

Selection Statements
Case Statement

```
case SCORE is
  when 85..100 => GRADE := 'A';
  when 75..84  => GRADE := 'B';
  when 60..74  => GRADE := 'C';
  when 0..59   => GRADE := 'F';
  when others  => null;
end case;

case GRADE is
  when 'A' | 'B' | 'C' => put_line ('pass');
  when 'F' => put_line ('fail');
  when others  => put_line ('invalid score');
end case;

case SELECTION_CODE is
  when 'W' =>
     BALANCE := BALANCE - AMOUNT;
     Put_Line ("Removing cash");
  when 'D' =>
     BALANCE := BALANCE + AMOUNT;
     Put_Line ("Adding cash");
  when others => Put_Line ("Invalid key");
end case;
```

LECTURE NUMBER: 037

TOPIC(S) FOR LECTURE:
Structured data types in Ada

INSTRUCTIONAL OBJECTIVE(S):

1. To learn the features of the Ada's structured data types.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)
We have already discussed Ada's scalar data types (integers, etc).

(Learning Label- Today we are going to learn ...)
Today we are going to look at Ada's composite data types.

CONTENTS:

1. Ada data structures L37OH1

   a. Arrays are composite data structures which contain a collection of components of the same data type L37OH2. Ada provides two types of arrays: constrained and unconstrained. Constrained arrays have their lower and upper bounds for their indices defined in the type declaration L37OH3. Unconstrained arrays define their index type in the declaration but do not define their lower and upper bounds for their indices. These bounds are defined in the object declaration L37OH4. This feature allows objects of different sizes to be created from the same data type; we have seen in a previous class how this language feature is helpful in reusability. The operations and I/O for arrays are the same as for Pascal. L37OH5. Ada provides attributes for arrays (**First**, **Last**, **Range**, **Length**) Use the definitions of L37OH6 and work through the exercise on L37OH7.

   b. **String** data type is a predefined unconstrained array of type **Character**. At object declaration, the size of the string is specified. The I/O provided by **Text_IO** was discussed previously. Ada provides the attributes **Value** and **Image** for strings. L37OH8, L37OH9, L37OH10

   c. Record data types are composite data structures which contain a collection of components of possibly different data types. Records are the same in Ada as in Pascal except for the ability to initialize values at type declaration and object declaration.

Variant records and discriminants are covered in the textbook but will not be discussed in class. L37OH11, L37OH12

PROCEDURE:

teaching method and media:

Lecture and overheads are the chief media for this lecture.

vocabulary introduced:

INSTRUCTIONAL MATERIALS:

overheads:

L37OH1	Ada data types
L37OH2	Array data types
L37OH3	Constrained arrays
L37OH4	Unconstrained arrays
L37OH5	Arrays
L37OH6	Array attributes
L37OH7	Examples of array attributes
L37OH8	String data type
L37OH9	String data type - operations and I/O
L37OH10	String attributes
L37OH11	Record data type
L37OH12	Record data type - accessing, operations, I/O

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

READING ASSIGNMENTS:
Benjamin Chapter 5 (pp. 39-50)

RELATED READINGS:
Booch Chapter 8 (pp. 115-124)
Booch(2) Chapter 6 (pp. 105-115)

Ada Data Types

- Scalar
  - Discrete
    - Integer (**Integer, Natural, Positive**)
    - Enumeration
      - Character (**Character**)
      - Boolean (**Boolean**)
  - Real
    - Floating point (**Float**)
    - Fixed point
- Composite
  - Arrays
    - Constrained
    - Unconstrained
    - Strings (**String**)
  - Records
- Access (i.e., pointers)
- Private
- Subtype and derived types

Array Data Types

Data structure consisting of a linear sequence of components of the same data type

Two types of arrays:

- Constrained
  - The lower and upper bounds for each array index are defined at type declaration
- Unconstrained
  - The lower and upper bounds for each array index are not defined at type declaration but are defined at object declaration

Constrained Arrays

Type declarations

```
type COST is array (1..8) of Float;
type MATRIX is array (1..3, 1..3) of Integer;
```

Object declarations

```
COAT_COSTS : COST;

DRESS_COSTS : COST := (90.0, 80.0,
    70.0, 60.0, 97.5, 81.0, 72.0, 85.0);

DRESS2_COSTS : constant COST := (90.0,
    80.0, 70.0, 60.0, 97.5, 81.0, 72.0, 85.0);

JEAN_COSTS : COST := (1 => 30.0,
    2 => 19.0, 3 => 15.50, 4 => 56.0,
    5 => 27.50, others => 28.0);

-- called anonymous array object
NEW_COSTS : array (1..8) of Float;

MATRIX_1, MATRIX_2 : MATRIX;
```

Unconstrained Arrays

Type declarations

```
type VECTOR_TYPE is array
    (Integer range <>) of Integer;

type MATRIX_TYPE is array
    (Positive range <>, Positive range <>)
    of Integer;
```

Object declarations

```
VECTOR1 : VECTOR_TYPE (1..30);

VECTOR2 : VECTOR_TYPE (1..10);

MATRIX1 : MATRIX_TYPE (1..3, 1..10);

MATRIX2 : MATRIX_TYPE (1..4, 1..6);
```

Arrays

Array I/O
    By component data type

Accessing arrays
    By component
        COAT_COSTS (2) := 124.50;
        Put (COAT_COSTS (3));

    As whole
        Can use :=, =, /=
        Must be same data type

        MATRIX_1 := MATRIX_2;

        Relational operators <, <=, >, and >= may be applied to one dimensional arrays whose elements are of a discrete type

    By slice
        Accessing consecutive components of an array

        MATRIX_1 (1..3) := MATRIX_2 (4..6);

Array Attributes

First	returns lower bound of first array index
First(N)	returns lower bound of Nth array index
Last	returns upper bound of last array index
Last(N)	returns upper bound of Nth array index
Range	returns range of first array index
Range(N)	
Length	returns number of elements in first index range
Length(N)	

Examples of Array Attributes

```
type MATRIX_TYPE is array (1..3, 0..30) of
Integer;

MATRIX : MATRIX_TYPE;


MATRIX_TYPE'First (1)

MATRIX_TYPE'First (2)

MATRIX_TYPE'First

MATRIX'Last (1)

MATRIX'Last (2)

MATRIX_TYPE'Range (1)

MATRIX_TYPE'Range (2)

MATRIX'Length
```

String Data Type

```
type STRING is array (Positive range <>)
    of Character;
```

Object declarations

```
FILE_NAME : String (1..30);

ERR_MSG   : constant String
            := "error -- file not found");

LINE      : String (1..80) := (others => "");
```

String Data Type

String operations
- Relational operators (=, /=, <, <=, >, >=)
- Assignment (:=)
- Concatenation (&)

String I/O

```
Put_Line (Item : in String);

Put (Item : in String);

Get_Line (Item : out String;
          Last : out Natural);

Get_Line (Item : out String);
```

String Attributes

Value	function that maps the value of **String** into the corresponding value of the discrete type

```
X : integer := Integer'Value ("1000");
```

Image	function that maps values of integer or enumeration type into an expression of type **String**

```
ONE_THOUSAND : String (1..4)
    := Integer'Image (X);
```

Record Data Type

Data structure consisting of a collection of components of the possibly different data types

Type declarations

```
type NAME_TYPE is
  record
    LAST_NAME      : String (1..20);
    FIRST_NAME     : String (1..20);
    MIDDLE_INITIAL : Character;
  end record;
```

Object declarations

```
MY_NAME : NAME_TYPE;

YOUR_NAME : NAME_TYPE :=
    (MIDDLE_INITIAL => ' ',
     others => "                    ");

HIS_NAME : NAME_TYPE :=
    ("Appleseedless       ",
     "Johnny              ", 'P');
```

Record Data Type

Accessing records

MY_NAME.MIDDLE_INITIAL := 'T';

Operations on records

:=, =, /=

Component selection

Record I/O

By component only

LECTURE NUMBER: 038

TOPIC(S) FOR LECTURE:
Access(pointer) data types in Ada

INSTRUCTIONAL OBJECTIVE(S):

1. To learn the features of Ada's access data types.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)
We have already discussed scalar and composite data types.
(Learning Label- Today we are going to learn ...)
Today we'll discuss access, or pointer data types.

CONTENTS:

1. Ada access data types L38OH1

   a. The access data type (often called pointer data type) enables the dynamic creation of objects during the execution of a program.

   L38OH2

   b. There are three steps to using access data types. First, the access type and objects of that type must be declared. When an object of an access data type is created, the object is automatically initialized to the value **null**. Here B1, B2, and B3 are initialized to null. This is the only case in which Ada defines an implicit default value.

      Second, objects of the access type are dynamically allocated. Using the allocator **new** in an assignment statement dynamically allocates the objects of an access type. The allocation process also allows for the initialization of values either by positional or named notation.

      Third, allocated objects can be referenced at execution time.

   L38OH3

   c. Access data types can be compared using the relational operators = or /=. This is valid only if they are of the same type.Ada also provides a notation **.all** which refers to all the values of an access data type (e.g., all the fields of the record type if the access data type pointed to a record).

L38OH4

d. Illustrate pointers through the linked list implementation of a tree as shown in L38OH4.

PROCEDURE:

teaching method and media:

Lecture and overheads are the chief media for this lecture.

vocabulary introduced:

INSTRUCTIONAL MATERIALS:

overheads:

L38OH1	Ada data types
L38OH2	Steps to using access types in Ada
L38OH3	Operations on access types
L38OH4	Example of linked list

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 028 - Detailed design review presentation

READING ASSIGNMENTS:

Benjamin Chapter 7 (pp. 63-72)

RELATED READINGS:

Booch Chapter 8 (pp. 124-129)
Booch(2) Chapter 6 (pp. 115-121)

Ada Data Types

- Scalar
  - Discrete
    - Integer (**Integer, Natural, Positive**)
    - Enumeration
      - Character (**Character**)
      - Boolean (**Boolean**)
  - Real
    - Floating point (**Float**)
    - Fixed point

- Composite
  - Arrays
    - Constrained
    - Unconstrained
    - Strings (**String**)
  - Records

- Access (i.e., pointers)

- Private

- Subtype and derived types

Steps to Using Access Types in Ada

1. Declare access type and objects of that type.

```
type BUFFER is
  record
    MESSAGE   : String (1..10);
    PRIORITY  : Integer;
  end record;
type BUFFER_PTR is access BUFFER;

B1, B2, B3  : BUFFER_PTR;
```

2. Dynamically allocate objects of the access type.

```
B1 := new BUFFER;
B2 := new BUFFER'(MESSAGE => "**********",
                  PRIORITY => 2);
```

3. Work with allocated objects.

```
B1 := null;
B1.MESSAGE := "Error - P1";
B1.PRIORITY := 10;
B2 := B1;
```

Operations on Access Types

Relational operators (=, /=)

Assignment (:=)

.all notation

Example of Linked List

```
type NODE;
type TREE is access NODE;
type NODE is
  record
    LEFT   : TREE;
    VALUE  : string (1..5);
    RIGHT  : TREE;
  end record;

ROOT  : TREE;
TEMP  : TREE;
PTR   : TREE;

ROOT := new NODE;
ROOT.VALUE := "NODE1";
TEMP := new NODE;
TEMP.VALUE := "NODE2";
ROOT.LEFT := TEMP;

TEMP := new NODE;
TEMP.VALUE := "NODE3";
PTR := ROOT.LEFT;
PTR.RIGHT := TEMP;
```

LECTURE NUMBER: 039

TOPIC(S) FOR LECTURE:
Procedures, functions, and packages in Ada

INSTRUCTIONAL OBJECTIVE(S):

1. To learn the features of the Ada programming language concerning procedures, functions, and packages.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)
In other lectures we have looked at using Ada specifications in high level design. The implementation details and the special capabilities of Ada, such as overloading, were not discussed.
(Learning Label- Today we are going to learn ...)
Today we shall revisit Ada's packages, functions, and procedures examining the details needed to implement these language structures..

CONTENTS:

1. Ada subprograms L39OH1

   a. As in other programming languages, procedures and functions are provided in Ada as fundamental tools for designing and building modular programs. Procedures are used to execute a group of statements, and functions are used to determine and return a value that is used in an expression.

   b. Ada allows for procedures and functions to have a specification part and a body or implementation part L39OH2. The specification part is the interface information and may be compiled in a file separate from the body. This separate compilation feature allows the subprogram to be used by other compilation units without the implementation details in the subprogram body being completed. Also if changes are later made to the implementation details, there is no need to recompile the other compilation units which use this subprogram.

   c. Ada provides three parameter modes L39OH3. The **in** mode allows the parameter to act as a local constant within the subprogram, the **out** mode allows the parameter to be assigned a value by the subprogram, and the **in out** mode allows the formal parameter to be initialized to the value of the actual parameter and then another value to be assigned to the parameter. The **in** mode is the default parameter mode.

d. For **in** parameters only L39OH4, a default value may be specified in the formal parameter list, and then the corresponding actual parameter may be omitted. If the default parameter is not the last parameter in the formal parameter list, named notation must be used to specify any remaining actual parameters in the procedure call.

e. Ada provides two types of parameter association L39OH5. In positional notation, which is the type commonly used in other programming languages, the order of the actual parameters must match the order of the formal parameters. In named notation, the name of the formal parameter is associated with the actual parameter so that the order of the actual parameters does not matter. Additionally, once named notation is used in a list, it must be used for the remaining parameters in the list.

f. The **return** statement can be used in both procedures and functions to terminate the subprogram and transfer control back to the calling routine L39OH6. In a function, the value to be returned is also indicated in the **return** statement. L39OH8 There can be one or more **return** statements in any subprogram, but there must be at least one in a function. A complete example showing the declaration, body, and invocation of a procedure and function are shown on L39OH7 and L39OH8.

g. Overloading is the use of the same name or operation symbol for different entities whose scope overlap. They are said to be overloaded provided that there is no ambiguity. Overloading can be used for subprogram identifiers as well as operators, task entry identifiers, and enumeration literals L39OH9. The compiler distinguishes the correct intended function by the parameters passed or by the context of an overloaded name. Therefore, for all overloaded subprograms, some aspect of the subprogram profile (the parameter order, number, or type or the returned data type on a function) must be unique from other subprograms with the same name. If some ambiguous reference is made, explicit resolution of the conflict can be made through named parameter association or a qualified expression L39OH10.

h. Discuss the scope and visibility rules L39OH11

2. Ada packages L39OH12

   a. Ada packages were introduced in an earlier class. We saw that packages provide a mechanism for organizing large or complicated programs.

   b. Private and limited private data types can be declared in a package specification L39OH13. These data types allow the name of the identifier to be visible but the implementation details of the data types to be hidden. These data types are used to restrict the operations available for certain data types. L39OH14, L39OH15

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

Lecture and overheads are the chief media for this lecture.

<u>vocabulary introduced</u>:

parameter modes
private data type
limited private data type

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>:

L39OH1	Ada program units
L39OH2	Ada procedures and functions
L39OH3	Parameter modes
L39OH4	Default parameter values
L39OH5	Parameter association
L39OH6	Return statement
L39OH7	Procedure
L39OH8	Function
L39OH9	Overloading
L39OH10	Explicit resolution with overloading
L39OH11	Scope and visibility rules
L39OH12	Packages
L39OH13	Private and limited private data types
L39OH14	Private data type
L39OH15	Limited private data type

<u>handouts</u>:

<u>RELATED LEARNING ACTIVITIES</u>:

(labs and exercises)
Lab 029 - Feedback on detailed design

__READING ASSIGNMENTS__:
Benjamin Chapter 6 and 8 (pp. 51-62 and 73-78)

__RELATED READINGS__:
Booch Chapter 13 (pp. 218-241)
Booch(2) Chapter 11 (pp. 216-240)

Ada Program Units

Procedures and functions

Packages

Tasks

Ada Procedures and Functions

Procedure

Used to execute a group of statements

Function

Used to determine and return a value that is used in an expression

Each has a specification (interface information) and body (implementation details)

Permits a subprogram name and parameter requirements to be available to other compilation units separate from the subprogram body

Parameter Modes

Indicates the flow of the data between the caller and the called unit

3 types of parameter mode:

1. **in**
   Acts as a local constant to the procedure

   ```
   procedure DRAW_LINE (FROM,
               TO : in POINT);
   ```

2. **out**
   A variable which may only be assigned a value by the procedure

   ```
   procedure FIND_MAX (A, B : in Integer;
               MAX  : out Integer);
   ```

3. **in out**
   A variable which is initialized to the value of the actual parameter and may be assigned a value by the procedure

   ```
   procedure SORT (DATA : in out
               NAMES);
   ```

Default Parameter Values (for IN only)

```
type DIRECTION is (ASCENDING,
DESCENDING);

procedure SORT (DATA  : in out NAMES;
    ORDER : in DIRECTION := ASCENDING);


SORT (CLASS, DESCENDING);

SORT (CLASS);  -- uses default value

-------------------------------------------------

procedure SPACES (COUNT : Integer := 1) is
begin
  for I in 1..COUNT loop
     Put (" ");
  end loop;
end SPACES;


SPACES (4);

SPACES; -- only one space will be output
```

Parameter Association

Positional notation
    Order of actual parameters must match order of formal parameters

Named notation
    Name of formal parameter is associated with actual parameter

```
procedure SEARCH_FILE (KEY : in NAME;
                       INDEX: out FILE_INDEX);
```

Positional

```
SEARCH_FILE ("SMITH J", RECORD_ENTRY);
```

Named Notation
3 variations with identical results

```
SEARCH_FILE (KEY => "SMITH J",
             INDEX => RECORD_ENTRY);

SEARCH_FILE (INDEX => RECORD_ENTRY,
             KEY => "SMITH J");

SEARCH_FILE ("SMITH J",
             INDEX => RECORD_ENTRY);
```

Return Statement

Used to terminate a procedure or function and transfer control back to the calling routine

In a function, the return statement indicates the value to be returned

```
procedure MIN (A, B : in Integer;
               C : out Integer) is
begin
  if (A < B) then
     C := A;
     return;
  end if;
  C := B;
end MIN;

function MIN (A, B : in Integer)
          return Integer is
begin
  if (A < B) then
     return A;
  end if;
  return B;
end MIN;
```

Procedure

```
procedure DIVIDE (DIVIDEND,
                  DIVISOR : in Integer;
                  QUOTIENT,
                  REMAINDER : out Integer);
```

```
procedure DIVIDE (DIVIDEND,
                  DIVISOR : in Integer;
                  QUOTIENT,
                  REMAINDER : out Integer) is
begin
   QUOTIENT := DIVIDEND / DIVISOR;
   REMAINDER := DIVIDEND rem DIVISOR;
end DIVIDE;
```

```
DIVIDE (120, 4, A_QUOTIENT,
        A_REMAINDER);
```

Function

```
function AVERAGE (A, B, C : in Float)
                        return Float;
```

```
function AVERAGE (A, B, C : in Float)
                        return Float is
   SUM  : Float;
begin
   SUM : A + B + C;
   return SUM / 3.0;
end AVERAGE;
```

```
CURRENT_AVERAGE := AVERAGE (TEST1,
                        TEST2, TEST3);
```

Overloading

The same name or operation symbol is used for different entities whose scope overlap and there is no ambiguity.

Allowable for:

- Operators
- Subprogram identifiers
- Task entry identifiers
- Enumeration literals

Meaning determined by operand, parameters, or context of use

Explicit Resolution with Overloading

2 means of explicit resolution in meaning for subprograms:

1. Named parameter association
2. Qualified expression

```
type BEEF is (STANDARD, GOOD, CHOICE,
              PRIME);
type INTEREST is (PRIME, BONDS,
              DISCOUNT);

procedure PROCESS (THE_CUT : BEEF);
procedure PROCESS (THE_RATE : INTEREST);

PROCESS (PRIME);  -- ambiguous reference

PROCESS (THE_RATE => PRIME);

PROCESS (BEEF'(PRIME));
```

Scope and Visibility Rules

3 types of visibility possible for an object:

1. Directly visible
2. Visible by selection
3. Hidden

```
procedure P1 is
   X   : Integer;
   Y   : Integer;
     procedure P2 is
           X : Integer;
     begin
           X := 4;  -- X of P2 directly visible
           P1.X := 3;  -- visible by selection
           Y := 5;  -- Y of P1 directly visible
     end P2;

begin -- P1
   P2;  -- directly visible
   X := 6;  -- X of P1 directly visible
   for I in 1..3 loop
       for I in 1..10 loop
           -- outer I is hidden in inner loop
       end loop;
   end loop;
end P1;  adapted from Benjamin
```

Packages

Programming unit that allows a collection of related entities to be made available for use by other program units

```
package STACKS is
   type STACK is private;
   procedure PUSH (ELEMENT : in integer;
                            ON : in out STACK);
   procedure POP (ELEMENT : out integer;
                            ON : in out STACK);
private
  -- STACK defined
end STACKS;


with STACKS; use STACKS;
procedure MAIN is
   A_STACK, B_STACK : STACK;
begin
   PUSH (10, A_STACK);
   PUSH (20, B_STACK);
end;
```

Private and Limited Private Data Types

Defined in a package specification

Name is visible to user but implementation details are hidden

Purpose: to restrict operations available outside of package body

Private Data Type

Operations available on private data types:

Assignment (:=)

Relational operators (=, /=)

Subprograms defined in its package

```
package STACKS is
  type STACK is private;
  procedure PUSH (ELEMENT : in integer;
                          ON : in out STACK);
  procedure POP (ELEMENT : out integer;
                          ON : in out STACK);
private
  MAX_ELEMENTS : constant integer := 100;
  type LIST is array (1..MAX_ELEMENTS) of
            integer;
  type STACK is
     record
        STRUCTURE : LIST;
        TOP : integer range 1..MAX_ELEMENTS;
     end record;
end STACKS;
```

Limited Private Data Type

Operations available on limited private data types:

Only subprograms defined in its package

```
package STACKS is
  type STACK is limited private;
  procedure PUSH (ELEMENT : in integer;
                  ON : in out STACK);
  procedure POP (ELEMENT : out integer;
                 ON : in out STACK);
private
  MAX_ELEMENTS : constant integer := 100;
  type LIST is array (1..MAX_ELEMENTS) of
           integer;
  type STACK is
     record
        STRUCTURE : LIST;
        TOP : integer range 1..MAX_ELEMENTS;
     end record;
end STACKS;
```

LECTURE NUMBER: 040

TOPIC(S) FOR LECTURE:
Generics in Ada

INSTRUCTIONAL OBJECTIVE(S):

1. To learn the features of Ada generics.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)
Often we find components of a system that are very similar in the task they perform (e.g., a sort procedure for an integer array, a sort procedure for a character array, and a sort procedure for a real array). Separate procedures, each of which uses the same sort algorithm, are written. A means of creating a template of a component (e.g., a sort template) that could be tailored at execution time would be beneficial. Ada generics provide such capability.

(Learning Label- Today we are going to learn ...)
Today we are going to examine the syntax for building generic program units.

CONTENTS:

1. Suppose you are asked to write a procedure to return the successor of a given element in a "days of the week" list. Suppose a further requirement is that the list be treated as a circular list (e.g., the successor of the last element is the first element). A simple solution is shown in L40OH1.

   L40OH2
   Three more generalized solutions are shown in this overhead. Each of these attempts at generality has drawbacks. The first addresses a logic problem by taking advantage of a runtime error; the second is not completely general because it "hard codes" the values of the first and last elements of the list; the third, while more general, has limited potential for reuse beyond this particular instance of a list.

2. L40OH3
   Ask how the third solution on L40OH2 could be modified to work for a list of integers from 1 to 10. Students are likely to quickly realize that defining a type SIZE as shown and substituting SIZE for DAYS in the function WRAP will work. Similarly ask how they could make this function work for the letters of the alphabet.

3. a. This method of achieving generality by changing the data type upon which the function operates is the way generic subprograms achieve generality.

   L40OH4 shows the generic for the WRAP function. Describe how type ELEMENT can be instantiated as DAYS, SIZE, or any other discrete data type.

   b. A generic program unit defines a template from which other kinds of program units can be created, as in L40OH4. For example, a generic subprogram may be used to create a class of similar program units whose only differences are based on the types upon which they operate. A subprogram or package may be made into a generic in Ada.

4. L40OH5

   a. Discuss the three aspects to defining and using a generic unit: the generic unit declaration, the generic subprogram or package body, and the generic instantiation L40OH5. The instantiation is the process of creating a program unit from a generic program unit. A generic program unit can't be called directly, but an instantiation of a generic program unit must be created. An instantiation is a declaration, not a statement and therefore must appear in the declarative part of a program unit which uses the function.

   b. L40OH6, L40OH7, L40OH8, L40OH9, L40OH10
   Use these overheads to illustrate generic subprograms, package bodies, and instantiations.

   c. One possible kind of parameter to a generic program unit is a data type. Generic formal parameter types determine the type of parameter with which the generic can be instantiated and the operations available on objects of that type within the generic body.

   L40OH11 shows the eight generic formal parameter types.

   d. Another possible kind of parameter to a generic program unit is an object and value. Discuss the example of this type of parameter as shown on overhead L40OH12.

   e. The last kind of parameter to a generic program unit is a subprogram. A generic formal subprogram matches with any actual subprogram having the same parameter and return-type profile. Examples are given on overheads L40OH13 and L40OH14.

PROCEDURE:

teaching method and media:

vocabulary introduced:

INSTRUCTIONAL MATERIALS:

overheads:

L40OH1	Moving towards generic units
L40OH2	Moving towards generic units
L40OH3	Moving towards generic units
L40OH4	Moving towards generic units
L40OH5	Ada generic unit
L40OH6	Generic subprograms
L40OH7	Generic subprograms
L40OH8	Specification of generic package
L40OH9	Body of generic package
L40OH10	Instantiation of generic package
L40OH11	Generic formal parameter types
L40OH12	Generic formal objects
L40OH13	Generic formal subprograms
L40OH14	Generic formal subprograms (cont.)

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 029 - Feedback on detailed design review presentation

READING ASSIGNMENTS:
Benjamin Chapter 9 (pp. 79-87)

RELATED READINGS:
Booch Chapter 14 (pp. 243-257)
Booch(2) Chapter 12 (pp.242-257)

Moving Towards Generic Units

```
type DAY is       (MONDAY, TUESDAY,
                  WEDNESDAY, THURSDAY,
                  FRIDAY, SATURDAY, SUNDAY);


function WRAP (D : DAYS) return DAYS is
begin
    if D = SUNDAY then
        return MONDAY;
    elsif D = MONDAY then
        return TUESDAY;
    elsif D = TUESDAY then
        return WEDNESDAY;
    elsif D = WEDNESDAY then
        return THURSDAY;
    elsif D = THURSDAY then
        return FRIDAY;
    elsif D = FRIDAY then
        return SATURDAY;
    else
        return SUNDAY;
    end if;
end WRAP;
```

Moving Towards Generic Units

```
-----------------------------------------------------------
function WRAP (D : DAYS) return DAYS is
begin
  return DAYS'Succ (D);
exception
  when Constraint_error =>
    return DAYS'First;
end WRAP;
-----------------------------------------------------------
```

```
-----------------------------------------------------------
function WRAP (D : DAYS) return DAYS is
begin
  if D = SUNDAY then
    return MONDAY;
  else
    return DAYS'Succ (D);
end WRAP;
-----------------------------------------------------------
```

```
-----------------------------------------------------------
function WRAP (D : DAYS) return DAYS is
begin
  if D = DAYS'Last then
    return DAYS'First;
  else
    return DAYS'succ (D);
end WRAP;
-----------------------------------------------------------
```

Moving Towards Generic Units

```
function WRAP (D : DAYS) return DAYS is
begin
  if D = DAYS'Last then
    return DAYS'First;
  else
    return DAYS'succ (D);
end WRAP;
```

What modifications would you make to **WRAP** in order to provide a wrap-around successor capability for the type **SIZE**?

```
type SIZE is range 1..10;
```

Moving Towards Generic Units

```
-- generic specification
generic
  type ELEMENT is (<>);
function WRAP_AROUND (D : ELEMENT)
                return ELEMENT;



-- generic body
function WRAP_AROUND (D : ELEMENT)
                return ELEMENT is
begin
  if D = ELEMENT'Last then
    return ELEMENT'First;
  else
    return ELEMENT'Succ (D);
  end if;
end WRAP_AROUND;


-- generic instantiation
function WRAP is new
    WRAP_AROUND (ELEMENT => DAYS);
function WRAP is new
    WRAP_AROUND (ELEMENT => SIZE);
function WRAP is new
    WRAP_AROUND (Character);
```

Ada Generic Unit

Defines a template from which other kinds of program units can be created

Subprogram or package can be a generic

Three aspects of defining and using a generic unit:

1. Generic unit declaration
2. Generic subprogram or package body
3. Generic instantiation

Generic Subprograms

A subprogram that will handle values of arbitrary type

```
procedure GENERIC_DEMO is
  X, Y  : Integer;
  generic
  -- generic specification
    type ELEM is private;
  procedure EXCHANGE (U, V : in out ELEM);
  -- generic body
  procedure EXCHANGE (U, V : in out ELEM) is
    T : ELEM;
  begin
    T := U;
    U := V;
    V := T;
  end EXCHANGE;
  -- end of generic declaration
   -- generic instantiation
  --       (goes in program that invokes SWAP)
  procedure SWAP is new EXCHANGE (Integer);
  -- generic subprogram call
    begin
      X := 1;
      y := 2;
      SWAP (X, Y);

end GENERIC_DEMO;
```

Generic Subprograms

```
------------------------------------------------------------
generic
-- generic specification
   type ELEM is private;
   with function "*" (LEFT, RIGHT : ELEM)
            return ELEM is <>;
function SQUARING (X : ELEM) return ELEM;
------------------------------------------------------------
-- generic body
function SQUARING (X : ELEM) return ELEM is
begin
   return X * X;
end SQUARING;
-- end of generic declaration
------------------------------------------------------------
-- example of use
with SQUARING;

-- instantiation of SQUARE
procedure FUNCTION_DEMO is
   function SQUARE is new SQUARING (Integer);
   X : Integer := 8;
begin
   X := SQUARE (X);
end FUNCTION_DEMO;
```

Specification of Generic Package, GENERIC_LIST

```
generic
   type ELEM is private;

package GENERIC_LIST is
   type CELL is private;
   type POINTER is access CELL;
   type ARR is array (Integer range <>) of ELEM;
   function MAKE (A : ARR) return POINTER;
   function FRONT (P : POINTER) return ELEM;
   function REST (P : POINTER) return POINTER;
   function ADD_ON (E : ELEM;
                    P : POINTER) return POINTER;

private
   type CELL is
      record
         VALUE : ELEM;
         LINK  : POINTER;
      end record;
end GENERIC_LIST;
```

Body of Generic Package, GENERIC_LIST

```
package body GENERIC_LIST is

   function MAKE (A : ARR) return POINTER is
      P : POINTER;
   begin
      for X in reverse A'Range loop
         P := ADD_ON (A(X), P);
      end loop;
      return P;
   end MAKE;

   function FRONT (P : POINTER) return ELEM is
   begin
     return P.Value;
   end FRONT;

   function REST (P : POINTER) return POINTER is
   begin
     return P.LINK;
   end REST;

   function ADD_ON (E : ELEM;
                    P : POINTER) return POINTER is
   begin
     return new CELL'(E,P);
   end ADD_ON;

end GENERIC_LIST;
```

Instantiation of Generic Package, GENERIC_LIST

```
type PERSON is
   record
      LAST_NAME : String (1..10);
      SSN       : SOC_SEC_NUM;
      BIRTHDAY  : DATE;
   end record;

package PERSON_LIST is
   new GENERIC_LIST (PERSON);

----------------------------------------------------------------

with GENERIC_LIST;

procedure LIST_DEMO is
   package INT_LIST is
      new GENERIC_LIST (ELEM => Integer);
   P : INT_LIST.POINTER;

begin
   P := INT_LIST.MAKE ((1, 2, 3, 4));
   P := INT_LIST.ADD_ON (5, P);
   while P /= null loop
      Put (INT_LIST.FRONT (P));
      P := INT_LIST.REST (P);
   end loop;
end LIST_DEMO;
```

Generic Formal Parameter Types

```
type ITEM is private;
        -- Matches almost any type
        -- Matches any type for which assignment and
        -- Tests for equality are available

type LIMITED_ITEM is limited private;
        -- Matches any type except for unconstrained
        -- Array type

type DISCRETE_ITEM is (<>);
        -- Matches any discrete type

type ARRAY_ITEM is array (DISCRETE_ITEM)
    OF ITEM;

type PTR_ITEM is access ARRAY_ITEM;

type INTEGER_ITEM is range <>;

type FLOAT_ITEM is digits <>;

type FIXED_ITEM is delta <>;
```

Generic Formal Objects

```
generic
   SIZE : Integer;  -- formal object
   type ELEM is private;
package STACK is
   procedure PUSH (E : ELEM);
   procedure POP return ELEM;
end STACK;

package body STACK is
   SPACE : array (1..SIZE) of ELEM;
   INDEX : SPACE'range := 1;
   procedure PUSH (E : ELEM) is
   begin
      SPACE (INDEX) := E;
      INDEX := INDEX + 1;
   end PUSH;
   procedure POP return ELEM is
   begin
      INDEX := INDEX - 1;
      return SPACE (INDEX);
   end;
end STACK;

-- instantiation
package INT_STACK is new STACK (25, Integer);
```

Generic Formal Subprograms

```
generic
  type ELEM is private;
  type VECTOR is array (Integer range <>)
       of ELEM;
  with function ">" (X, Y : ELEM) return Boolean;
procedure SORT (A : in out VECTOR);

------------------------------------------------------------------

with SORT;
procedure GENERIC_DEMO is
  type INT_VECTOR is array (Integer range <>)
       of Integer;
  A : INT_VECTOR (1..10) := (4, 7, 10, 2, 5, 8,
                             1, 2, 6, 9);
  procedure INT_SORT is new SORT (Integer,
                         INT_VECTOR, ">");

-- instantiation
begin
  INT_SORT (A);
  for X in A'range loop
     Put (X, 4);
  end loop;
end GENERIC_DEMO;
```

Generic Formal Subprograms

```
generic
  type ELEM is private;
  type VECTOR is array (Integer range <>)
       of ELEM;
  with function ">" (X, Y : ELEM)
       return boolean is <>;
procedure SORT (A : in out VECTOR);

--------------------------------------------------------------

type EMPLOYEE is
  record
    NAME : String (1..40);
    ID   : Integer;
  end record;
type EMPLOYEE_ARRAY is array (Integer
      range <>) of EMPLOYEE;
function GT (A, B : EMPLOYEE) return Boolean is
begin
  return A.ID > B.ID;
end GT;

-- instantiation
procedure EMPLOYEE_SORT is new SORT
     (EMPLOYEE, EMPLOYEE_ARRAY, GT);
```

LECTURE NUMBER: 041

TOPIC(S) FOR LECTURE:
Exceptions and exception handlers in Ada

INSTRUCTIONAL OBJECTIVE(S):

1. To learn the features of the Ada capabilities regarding exceptions and exception handlers.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)
Ada provides a mechanism for responding to and managing errors.

(Learning Label- Today we are going to learn ...)
Today we are going to learn about these capabilities and how to use them.

CONTENTS:

1. Ada exceptions

   a. Exceptions are error conditions that may arise during program execution and cause the suspension of normal program execution L41OH1. Common examples include division by zero and writing to a file that is not open. Discuss other types of error conditions. Real-time systems must have the ability to handle error situations to be reliable; termination of a program upon encountering an error is not always desirable and, in some cases, is disastrous. For example a pacemaker.

   b. Raising an exception brings the exception or error situation to attention of the programmer and it automatically responds by transferring control to an exception handler. Predefined exceptions are automatically raised by Ada; user-defined exceptions are raised by the **raise** statement L41OH2. The scope of user-defined exceptions is the same as identifiers.

2. Ada exception handlers

   a. An exception handler is that portion of code which responds to an exception L41OH3. An exception handler allows a program to recover from an error or, at a minimum, print a meaningful error message and terminate the program gracefully.

b. Exception handlers begin with the word exception and may appear at the end of a **begin..end** block or frame. This means that an exception handler can

appear at the end of the body of a subprogram, package, task and generic unit.

3. Exception propagation

a. If no exception handler is included in a unit, raising the exception causes several things to happen. First, execution of the unit is terminated and the exception is "propagated" to a unit at the next highest level that does contain an appropriate handler L41OH4. How the exception is propagated depends on the type of frame in which it was raised.

Discuss each of the results of an exception being raised in each of the program units in which an exception can be defined. For the frames in L41OH4, each frame terminates and then: for a block an exception is re-raised at the point immediately after the block, for a subprogram the same exception is raised at the point immediately following the subprogram call;for a package which is a library unit the main program dies otherwise the exception is raised in the next highest level; and a task merely terminates without re-raising the exception. Discuss the example in L41OH5

b. Special form of **raise** statement can be used inside an exception handler to re-raise the exception currently being handled to propagate the exception to the next higher level. See example of RAISE used to propagate an exception. L41OH6

c. Discuss User defined exceptions L41OH7. Trace the example in L41OH8

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

<u>vocabulary introduced</u>:

INSTRUCTIONAL MATERIALS:

overheads:

L41OH1	Ada exceptions
L41OH2	Predefined exceptions
L41OH3	Exception handlers
L41OH4	Exception propagation
L41OH5	Example of exception propagation
L41OH6	Special use of Raise statement
L41OH7	User-defined exceptions
L41OH8	Example of user-defined exception

handouts:

RELATED LEARNING ACTIVITIES:

(labs and exercises)

Lab 030 - Video on Code inspections

READING ASSIGNMENTS:

Benjamin Chapter 12 (pp. 111-117)

RELATED READINGS:

Booch Chapter 17 (pp. 312-322)

Booch(2) Chapter 15 (pp.318-335)

Ada Exceptions

Exception

An error condition which may arise during program execution and cause suspension of normal program execution

Exception handler

A portion of program text specifying a response to an exception

Allows program to recover from an error or, at a minimum, print a meaningful error message and terminate the program gracefully

Raising an exception

Brings the exception condition to the programmer's attention

Causes transfer of control to an exception handler

raise STACK_OVERFLOW;

Predefined Exceptions

Predefined by the language sp [illegible].cation

Automatically reported or raised in Ada programs

Examples:

Numeric_error	Attempt to divide by zero or occurrence of numeric overflow on a numeric operation such as addition
Constraint_error	Attempt to violate some form of constraint, including range constraints, index constraints, or discriminant constraints
Storage_error	Insufficient storage available to satisfy the run-time requirements of a program

Exception Handlers

Specifies the exception to be handled and the action to be taken for each exception

Only appear at the end of four different program units:

- A **begin..end** block
- Body of a subprogram
- Body of a package or generic unit
- Body of a task

```
with Text_IO; use Text_IO;
procedure EXCEPTION_DEMO is
   X : Integer range 1..20;
 begin
   X := 43;
   -- other statements

   exception
      when Constraint_error =>
            Put ("Constraint Error Exception ");
      when others =>
            Put ("Other Exception ");
 end EXCEPTION_DEMO;
```

Exception Propagation

When an exception is raised in a unit that does not define an exception handler for that exception, execution of the unit is terminated and the exception is propagated to a unit that does contain the appropriate handler

How exception is propagated depends on where it was raised

A **begin..end** block

Body of a subprogram

Body of a package

Body of a task

Example of Exception Propagation

```
with Text_IO; use Text_IO;

procedure EXCEPT_DEMO is
  procedure RAISE_EXC is
  begin
    Put ("   raise_exc");
    raise Constraint_error;
  end RAISE_EXC;

  procedure HANDLE_EXC is
  begin
      Put ("   handle_exc");
      RAISE_EXC;
   exception
     when Constraint_error =>
       Put ("   Constraint error caught");
  end HANDLE_EXC;

begin
  Put ("Except_demo: calling handle_exc");
  HANDLE_EXC;
  Put ("Except_demo: back from handle_exc");

 exception
  when Constraint_error =>
    Put ("Constraint error in Except_demo");

end EXCEPT_DEMO;
```

Special Use of Raise Statement

Special form of **raise** statement can be used inside an exception handler to re-raise the exception currently being handled

```
begin
  -- allocate some resource which shouldn't
  -- be permanently allocated; causes exception
exception
  when others =>
      -- code might clean up resources that were
      -- allocated in the enclosing unit
      raise;
end;
```

User-defined Exceptions

Programmers may define their own exceptions

Examples of exception declarations:

```
    TABLE_FULL     : exception;
    ILLEGAL_DATA   : exception;
    STACK_OVERFLOW : exception;
```

Scope of exception name is same as scope of other identifiers in a declaration

Example of User-defined Exception

```
with Text_IO; use Text_IO;
procedure SCOPE_DEMO is
  E : exception;

  procedure P is
  begin
    raise E;
  end P;

  procedure Q is
    E : exception;
  begin
    P;
  exception
    when E =>
      Put_line ("Handler for Q.E");
  end Q;

begin
  Q; -- raise exception handler in Q
  P; -- raise exception handler below

  exception
   when E =>
     Put_line ("Handler for Scope_Demo.E");
end SCOPE_DEMO;
```

LECTURE NUMBER: 042

TOPIC(S) FOR LECTURE:
Sequential and direct files in Ada

INSTRUCTIONAL OBJECTIVE(S):

1. To learn the Ada features regarding sequential and direct files.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

When we first began looking at the syntax of Ada, we examined the capabilities for files as provided in **Text_IO**. **Text_IO** works on text files which are treated as a stream of characters including end of line terminators, end of page terminators, and end of file terminator.

(Learning Label- Today we are going to learn ...)

Today we are examining how Ada supports sequential and direct files.

CONTENTS:

1. Ada files

   a. L42OH1
   Ada provides three packages for file services **Text_IO** which supports text files, **Sequential_IO** which supports sequential files, and **Direct_IO** which supports direct access files.

   b. **Sequential_IO** and **Direct_IO** are generic packages and, unlike TEXT_IO, must be instantiated. They provide the same data type (**File_Type**) as **Text_IO** and the same file modes. **Direct_IO** also provides another file mode **Inout_file** which allows for a read-write file. Both files provide the same procedures as **Text_IO** for creating, deleting, opening, closing, and resetting files.

   c. L42OH2
   Sequential_IO creates a sequential binary file of the same data type. Three additional subprograms are provided by this package for the support of sequential files: **Read**, **Write**, and **End_of_file**.

   d. L42OH3

In **Direct_IO**, files are viewed as a set of elements occupying consecutive positions. An element in the file can be randomly accessed and updated by its index which indicates its position in the file. The index numbering for a file starts at 1. An open direct access file maintains a current index which is the index of the component used in the next read or write operation. The following additional subprograms are provided in **Direct_IO**: **Read, Write, Set_Index, Index, Size**, and **End_of_File**

e. L42OH4
The subprograms for reading and writing can work off the current index or an index can be specified in the parameter list. If an index is specified it becomes the current index. For both operations, the current index is incremented by one after the operation is done.

f. L42OH5
Discuss the example of sequential I/O shown.

PROCEDURE:

teaching method and media:

Lecture and overheads are the chief media for this lecture.

vocabulary introduced:

INSTRUCTIONAL MATERIALS:

overheads:

L42OH1	Ada files
L42OH2	Sequential files
L42OH3	Direct-access files
L42OH4	Direct-access files (cont.)
L42OH5	Example of usefulness of Sequential_IO

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

READING ASSIGNMENTS:
Benjamin Chapter 10 (pp. 89-96)

RELATED READINGS:
Booch Chapter 19 (pp. 356-373)

Ada Files

Ada provides three packages for file services:

Text_IO	A package providing support for text files
Sequential_IO	A generic package providing support for sequential files
Direct_IO	A generic package providing support for direct-access files Uses another file mode **Inout_File**

Sequential Files

Subprograms provided in **Sequential_IO**:

```
procedure Read (File : in File_Type;
                Item : out Element_Type);


procedure Write (File : in File_Type;
                 Item : in Element_Type);


function End_of_File (File : in File_Type)
    return Boolean;
```

Direct-Access Files

File is viewed as a set of elements occupying consecutive positions; an element at arbitrary position can be randomly accessed and updated by its index

Numbering of index starts at 1

Subprograms provided in **Direct_IO**:

```
procedure Read (File : in File_Type;
                Item : out Element_Type);

procedure Read (File : in File_Type;
                Item : out Element_Type;
                From : in Positive_Count);

procedure Write (File : in File_Type;
                 Item : in Element_Type);

procedure Write (File : in File_Type;
                 Item : in Element_Type;
                 To   : in Positive_Count);
```

Direct-Access Files (cont.)

More subprograms provided in **Direct_IO**:

```
procedure Set_Index (File : in File_Type;
                     To   : in Positive_Count);

function Index (File : in File_Type)
                    return Positive_count;

function Size  (File : in File_Type)
                    return Count;

function End_of_File (File : in File_Type)
                    return Boolean;
```

Example of Usefulness of Sequential_IO

```
generic
   type ITEM is private;
   with function "<" (LEFT, RIGHT : ITEM)
           return Boolean is <>;
procedure GENERIC_MERGE_SORT
           (INPUT_FILE_NAME,
            OUTPUT_FILE_NAME : in String);

----------------------------------------------------------------

with Sequential_IO;
procedure GENERIC_MERGE_SORT
           (INPUT_FILE_NAME,
            OUTPUT_FILE_NAME : in String) is

      package ITEM_IO is new Sequential_IO
(ITEM);
   use ITEM_IO;

   -- rest of program

end GENERIC_MERGE_SORT;
```

<u>LECTURE NUMBER</u>: 043

<u>TOPIC(S) FOR LECTURE</u>:
tasks in Ada

<u>INSTRUCTIONAL OBJECTIVE(S)</u>:

1. To learn the features of Ada tasks.

<u>SET UP, WARM-UP</u>:
(How involve learner: recall, review, relate)
L43OH1 We have discussed several Ada program units. Tasks are another type of Ada program unit. They provide parallel processing. We will find that although they resemble packages, there are several significant differences between tasks and packages.

(Learning Label- Today we are going to learn ...)

<u>CONTENTS</u>:

1. Ada tasks
   L43OH2
   a. Tasks are the programming unit in Ada which provides parallel processing; in other words, a task can be activated and executed concurrently with other program units. A task is not itself a compilation unit but must be declared within a subprogram, package, or generic package. A task has a specification and a body. A task is declared in the declarative part of these programming units. A task is activated when the **begin** statement of its parent unit is reached. A task may be called by any other programming unit.

   b. A task can perform functions such as mutual exclusion and synchronization, which had been limited to operating systems.

   L43OH3
   c. The entry declaration in the task specification defines the functions or services that the task provides. The entry declaration is similar to a subprogram declaration.

   L43OH4
   d. The task body contains the accept statement which corresponds to the entry in the specification.

L43OH5

e. A rendezvous is the meeting of the calling unit and called task. This is an indivisible action where the calling unit and called task are locked together.

L43OH6

f. The stages of the calling unit and called task are shown on overheads L43OH7.

g. Use the program in lecture 16 to reinforce what students have learned about Ada.

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

Lecture and overheads are the chief media for this lecture.

<u>vocabulary introduced</u>:

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>:

L43OH1	Ada program units
L43OH2	Ada tasks
L43OH3	Task specification
L43OH4	Task body
L43OH5	Rendezvous
L43OH6	Stages of a rendezvous (entry call first)
L43OH7	Stages of a rendezvous (accept first)

<u>handouts</u>:

<u>RELATED LEARNING ACTIVITIES</u>:
(labs and exercises)

<u>READING ASSIGNMENTS</u>:

Benjamin Chapter 11 (pp. 97-109)

<u>RELATED READINGS</u>:

Booch Chapter 16 (pp. 276-309)
Booch(2) Chapter 14 (pp.280-315)

Ada Program Units

Procedures and functions

Packages

Tasks

Ada Tasks

Ada program unit that can be activated and executed concurrently with other program units

Allows certain capabilities previously performed only by operating system to be performed in language

Not a compilation unit; declared within a subprogram, package, or generic package

Has specification and body

Aspects involved in a task:

- Task entry
- Entry call
- Accept statement
- Rendezvous

Task Specification

Defines interface which other (related) program components use to interact with the task

Interface consists of task entry declarations

- Like the subprogram declarations in a package specification

- Define functions or services that the task provides

```
task SINGLE_TELLER is
  entry DEPOSIT (ID   : CUST_ID;
                 VAL  : MONEY;
                 STAT : out STATUS);
  entry WITHDRAW (ID   : CUST_ID;
                  VAL  : MONEY;
                  STAT : out STATUS);
  entry BALANCE (ID   : CUST_ID;
                 VAL  : out MONEY;
                 STAT : out STATUS);
end SINGLE_TELLER;
-----------------------------------------------
-- entry call to above task
SINGLE_TELLER.DEPOSIT   (ID,   AMOUNT,
STAT);
```

Task Body

```
task SINGLE_TELLER is
   function RANDOM_TRANSACTION_TIME is
   begin
     loop
        select
           accept DEPOSIT (ID   : CUST_ID;
             VAL  : MONEY;
             STAT : out STATUS) do
           BANK_DATABASE.VERIFY_CUST_ID
             (ID, VALID);
           if not VALID then
             STAT := BAD_CUST_ID;
           else
             BANK_DATABASE.DEPOSIT (ID, VAL);
             STAT := SUCCESS;
             end if;
           end DEPOSIT;
        or
           accept WITHDRAW (...) do..end;
        or
           accept BALANCE (...) do..end;
         end select;
      end loop;
end SINGLE_TELLER;
```

Rendezvous

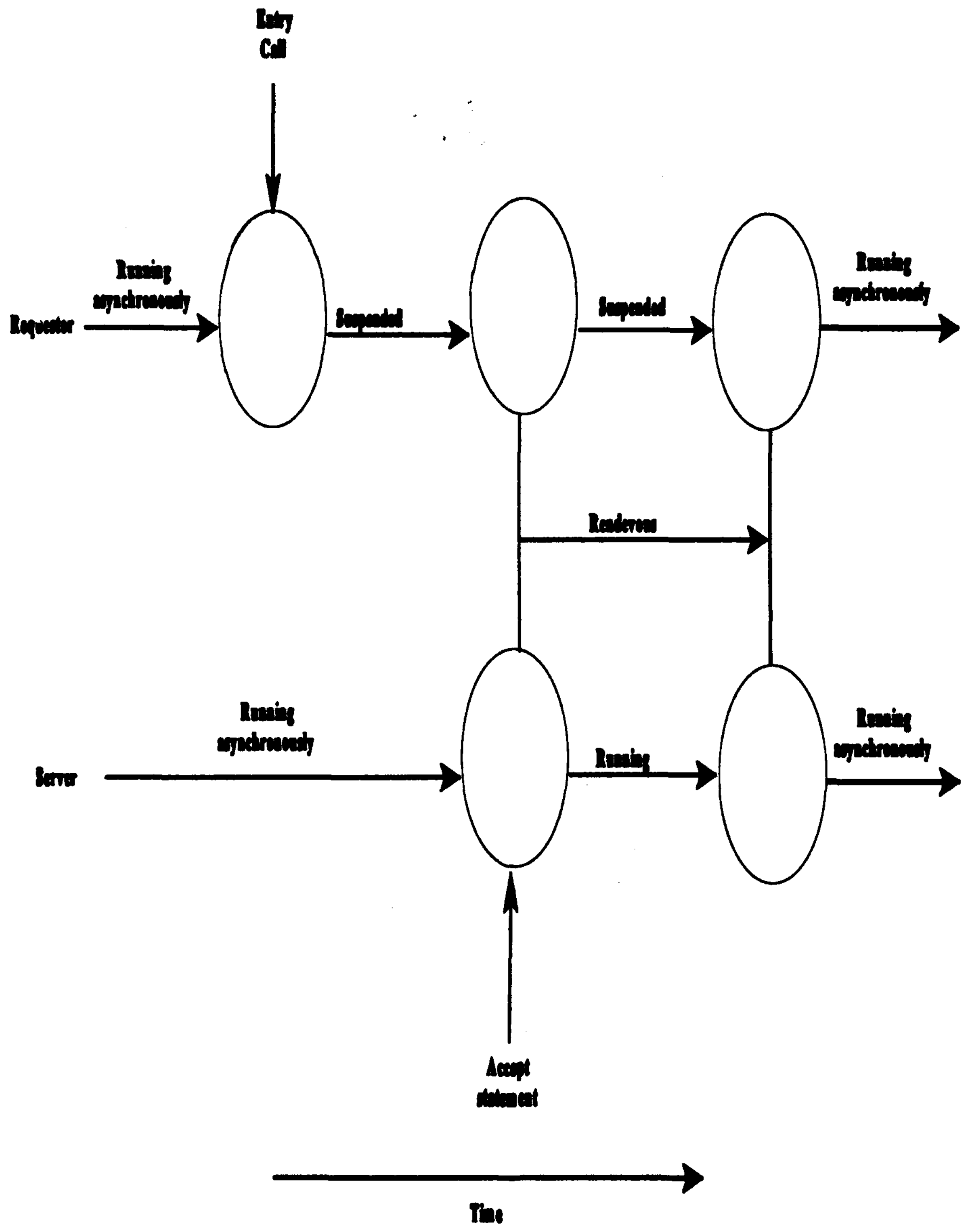
Meeting of calling and called tasks

Indivisible action

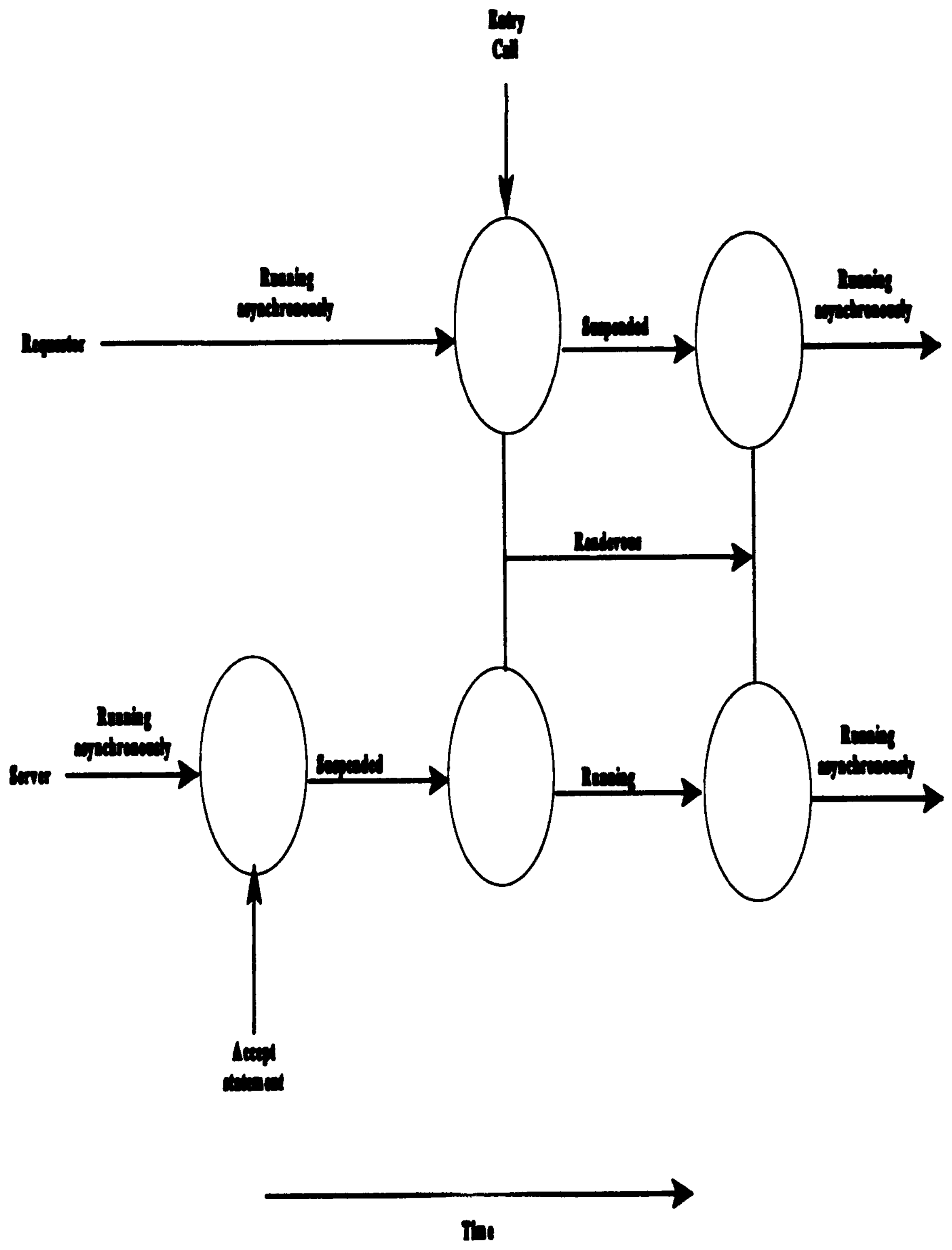
Two tasks locked together; calling task waits while called task executes; after called task completes, both tasks proceed independently of each other

Achieves:
- Synchronization
- Exchange of information
- Mutual exclusion

Stages of a Rendevous
Entry Call First



Stage of a Rendezvous Accept First



LECTURE NUMBER: 046

TOPIC(S) FOR LECTURE:
Introduction to use cases

INSTRUCTIONAL OBJECTIVE(S):

1. Understand Jacobson's concept of use cases and their use as an analysis, design, and test tool.
2. Be able to develop use cases.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)
We have discussed a number of ways to elicit requirements from the customer and user. The ability for a software developer to consider multiple viewpoints of the system and to consider a system from an external (to the system) is crucial. The requirements for a system describe the desired external behaviors of the system. A methodology has been developed recently which emphasizes the external behavior of a system as it relates to the system users.

(Learning Label- Today we are going to learn ...)
Today we're going to describe this method. It was developed by Jacobson and is called the use case driven approach.

CONTENTS:

1. Introduce Jacobson's use case concept. Use cases are one part of Jacobson's requirements model. The other components are a problem domain object model and a user interface model.

   Use cases are developed early so that the requirements, user interface, and test teams can get another view of the system.

2. L46OH1
   a. Jacobson's use case model involves actors and use cases as tools to identify/define what exists outside the system (actors) and what should be performed by the system (use cases). The actors include the users and user roles. Consider the Koff system. Who are the actors?

      (1) Customer (2) Operator (3) Owner

      Contrast how these actors interact with the system. Among other things, the customer removes tapes and returns tapes; the operator maintains the machine and stocks tapes; the owner gets reports and determines rental and sale tapes.

b. Use cases represent what the users should be able to do with the system. Each use case is a complete course of events initiated by an actor and it specifies the interaction that takes place between the actor and the system. Each time a user uses the system, he/she will perform a behaviorally related sequence of transactions in a dialogue with the system. Each of these is a use case, or a scenarios. A detailed description is written for each use case.

3. a. L46OH2
Jacobson discusses a recycling system as a first example. The system consists of a machine that accepts a variety of recyclable materials deposited by a users. Once a user deposits items, the machine generates a receipt based on the items deposited.

b. L46OH3
The customer should be able to deposit items. This forms one use case, Deposit Item. Discuss it.

c. L46OH3
The operator should be able to get a daily report of what items have been deposited. This is another use case, Generate Daily Report. Discuss it.

4. L46OH4
As requirements analysis continues and more information is determined, the use cases will be described in more detail. Discuss this elaboration of the Deposit Item use case.

5. Introduce the concept of "extends" with use cases. One use case can be inserted into another, thus extending the other use case. This is particularly useful in considering abnormal conditions.

L46OH5
Discuss Item Is Stuck as an example of this. Note that Deposit Item is described completely independent of this new use case.

6. Work through a use case, called Withdraw Tape, for a customer withdrawal of a rental tape in the Koff system. See if anyone notices that a receipt is never issued to the customer. Discuss this oversight.

7. **Pick any of the use case examples and discuss test cases that could be developed for it. Point out that the use case approach has application in both structured analysis and object-oriented analysis.**

PROCEDURE:

teaching method and media:

vocabulary introduced:

use case
scenario
actor

INSTRUCTIONAL MATERIALS:

overheads:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 032	Code Inspections

READING ASSIGNMENTS:

RELATED READINGS:

Jacobson Chapter 7 (pp. 148-195)

Jacobson's Use Case Model

Uses actors and use cases as tools to identify:

what exists outside the system (actors); and

what should be performed by the system (use cases).

Each time a user uses the system, he/she will perform a behaviorally related sequence of transactions in a dialogue with the system. Each of these is a use case, or a scenarios. A detailed description is written for each use case.

Deposit Item Use Case

Deposit Item is started by Customer when he/she wants to return cans, plastic containers, or glass containers. With each item that Customer places in the recycling machine, the system will increase the received number of items from Customer as well as the daily total of this particular type. When Customer has turned in all the items, he/she will press the receipt button to get a receipt containing a summary of the deposited items and the amount due.

Generate Daily Report Use Case

Generate Daily Report is started by Operator when he/she wants to print out information about the items deposited that day. The system will print out how many of each deposit items have been received this day, as well as the overall total for the day. The total number will be reset to zero to start a new daily report.

More Detailed Deposit Item Use Case

The course of events starts when the customer presses the "start-button" on the customer panel.

The customer can now deposit items via the customer panel. The sensors inform the system that an object has been inserted. They also determine the size and type of the deposit item and return these results to the system.

The daily total for the deposited item is incremented, as is the number of deposited items of that type from this customer.

When the customer is finished depositing items, he/she asks for a receipt by pressing the "receipt button" on the customer panel.

The system compiles the information to be printed on the receipt. For each type of deposit item, its return value and the number of deposited items from this customer is extracted.

The information is printed, with a new line for each deposit item, by the receipt printer.

Finally, the grand total for all returned deposit items is extracted by the system and printed out by the receipt printer.

Extending a Use Case

Item Is Stuck Use Case

Item Is Stuck is inserted into Deposit Item when Customer deposits an item that gets stuck in the recycling machine. Operator is called and Customer cannot turn in more items until Operator informs him/her that the machine can be used again.

LECTURE NUMBER: 047

TOPIC(S) FOR LECTURE:
Implementation languages

INSTRUCTIONAL OBJECTIVE(S):

1. Understand the factors involved in selecting a programming language.
2. Understand factors that affect the quality of source code.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

All of you have experience in several programming languages (Pascal from CS-1 and CS-2, assembly language, and Ada in this class). Some of you also know other languages (COBOL, C, FORTRAN).

(Learning Label- Today we are going to learn ...)

Today we're not going to look at any language in particular, but rather look at some factors that should be considered in evaluating languages. As we do this, think of the languages that you know and consider how aspects of the language are supportive of software engineering principles and how other aspects are not. You may have thought of a language strictly as an implementation tool. This was likely the case in CS-1.

At this point, however, you should be able to look beyond that. For example, Mynatt's detailed design checklist includes "Is the design appropriate for the target programming language?" Similarly, in our discussions of your project, the question of language support, i.e. how easily a design is implementable, has come up.

CONTENTS:

NOTE: This lecture is based on Mynatt's discussion of choosing a programming language in section 5.7. She offers many good examples and observations to support the ideas outlined here.

1. Ask students about the impact of language choice. Is the choice of language important? Why?

   Make sure student responses include significant impact the language will have on such things as:
   i Maintainability,
   ii Coding and testing,
   iii The amount of distortion between the implementation model and the design model,

iv the effectiveness of the personnel.

L47OH1
Note that good code can be written in any language, even assembly language, but it is much easier if the language itself actively supports the production of efficient, reliable, maintainable software. Point out that there are trade-offs involved and that no single language can simultaneously achieve a high degree of success in each of these areas. For example, Ada is highly maintainable but has low performance in terms of execution speed until properly tuned. On the other hand, C is difficult to maintain but has a high level of performance in terms of execution speed.

Ask what is the "language of choice" currently? In terms of the number of lines of code in existence and still being maintained, the answer is COBOL; the vast majority of software world-wide is written in COBOL.

2. L47OH2 - Pragmatic factors in choosing a language.
Point out that the question of what language is going to be used to implement a project does not even arise in many cases. Why? Because more often than not the choice is dictated by the by the customer or the organization or some standard, or something else beyond the particular project. What are some of these pragmatic factors?

a. Dictated by client. The client wants it written in BASIC, or COBOL, or Ada, for example. There are many reasons for such a requirement by the client, including organizational or sponsor standards that call for a particular language (e.g. DoD and Ada), availability of compiler (and/or other software development tools).

b. Existing expertise. Selecting a new language over one in which the coders are experienced involves training. The training needs to be sufficient to take advantage of the features of the new language in order to have the language used effectively.

Concerning the question of which language is "most suitable" for a given project, work through Schach's example involving the QQQ corporation (pages 340-341).

c. Language used in other projects, both previous and concurrent. Cite DoD problems that led to development of Ada.

d. Concurrency

3. L47OH3
Language characteristics that can support efficient, reliable, maintainable software. First ask class to list some characteristics.

a. Modularity - A language should support the partitioning of a large product into modules. To do so it must support separate compilation of modules.

   Note Ada's compilation units (procedures, functions, packages, generics, and tasks), each of which consists of two parts (the specification and the body) which are themselves separately compiled.

b. Power and suitability - Power refers to a language's ability to carry out programming tasks simply an with little programming effort. Power also is relative to the type of problem being solved. Different languages are suitable for different purposes.

c. Simplicity, clarity, orthogonality

   i Simplicity refers to the size of its vocabulary. Some languages are so large that it is difficult to become familiar with the entire language. This can result in a tendency for subsets to be adopted which, in turn, leads to incompatible subsets. Discuss Pascal as simple, PL/I and Ada as complex.
   ii Clarity refers to how natural, meaningful, and unambiguous the language is to the programmer. Some languages are more machine/architecture oriented and thus less problem oriented (and less clear).
   iii Orthogonality of a language refers to its consistency in allowing language features to be combined. It relates to clarity and simplicity because lack of orthogonality results in lots of special cases for the programmer to remember. Examples include Pascal's restrictions on reading/writing certain types.

d. Syntax - It is desirable that the syntax be simple, consistent, and supportive of clear code. Examples include:
   i method of indicating blocks (begin-end, etc). Note the explicit keyword approach of some languages (if-endif) versus the begin-end approach of others.
   ii format of statements - can encourage or discourage use of white space, indentation.
   iii rules for identifier names - can encourage or discourage use of meaningful identifiers.

e. Structured programming and control structures - An accepted definition of structured programming includes:

   i selection and iteration control structures with controlled goto (forward direction only, heavily restricted, perhaps to exception handling);

   ii one entry, one exit

   A language should have a strong effective implementation of the basic control structures.

f. Exception handling - Most programming languages do not include facilities to detect and handle exceptions. This is a serious defect.

g. L47OH4 - Typing
   According to Sommerville, "it is essential that a high-level programming language with strict typing be used for system development. Achieving fault-free software is virtually impossible if low-level programming languages with limited type checking are used."

   Discuss the "need to know" principle. This should be used to control access to system data. Key to effective information hiding is a language's typing system. Cite Ada's use of private types to ensure that the details of a data structure's implementation is inaccessible beyond where it is defined.

h. L47OH5 - Information hiding
   Languages like Ada and C++ offer direct support for information hiding. Refer to Sommerville section 15.1.2 for an excellent discussion of data typing.

i. L47OH6 - Procedural and data abstraction
   According to Mynatt, a language should include these features

   i Mechanisms for high-level encapsulation of procedural abstractions <u>and</u> data abstractions.

   ii Distinction between the specification of an abstraction and the implementation of an abstraction.

   iii Mechanisms to protect outside access to encapsulated information.

   iv Methods for importing modules from other sources.

j. Tools - structure editors, debuggers, programming environments, package libraries

k. Maintenance

l. Reuse (libraries, packages)

m. The level of a language's support for object-oriented approaches (e.g., direct support for the definition of classes, inheritance, encapsulation, and messaging) is becoming increasingly important.

Note the ongoing debate between proponents of Ada and C++. On the Ada side, it has strong typing that leads to greater reliability and maintainability and it is standardized which leads to true portability. On the C++ side, it is object-oriented and that should lead to ease of development and reuse.

4. L47OH7
Discuss the following factors affecting the quality of source code.

a. Use of structured coding techniques.

b. Good coding style.
   i Shorter is simpler.
   ii Fewer decisions is simpler.
   iii Nested logic should be avoided.
   iv Negative logic should be avoided.

c. Well-chosen local data structures.

d. L47OH3 - Well-written internal comments.
   i Program headers.
   ii Internal module headers.
   iii Line comments.

e. Readable, consistent source code format and identifier naming.
   i Vertical white space.
   ii Horizontal white space (indentation).
   iii Readability of comments (spelling, grammar, clarity)

f. *Summarize* the discussion by discussing how critical the adoption of a written set of programming standards is to enforcing quality code. Call attention to the Ada Quality and Style manual.

g. Point out also that well designed modules (highly cohesive, loosely coupled) are easier to document. Ask why and discuss.

PROCEDURE:

teaching method and media:

vocabulary introduced:

structured programming
syntax
orthogonality

INSTRUCTIONAL MATERIALS:

overheads:

L47OH1	Active support from implementation language
L47OH2	Pragmatic factors in language selection
L47OH3	Features language should/must support/possess - 1
L47OH4	The case for strong typing
L47OH5	Features language should/must support/possess - 2
L47OH6	Mynatt: procedural and data abstraction
L47OH7	Factors affecting quality of source code - 1
L47OH8	Factors affecting quality of source code - 2

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

READING ASSIGNMENTS:

Mynatt Chapter 5 (pp. 207-235)
Mynatt Chapter 6 (pp. 239-271)

RELATED READINGS:

Pressman Chapter16 (pp. 513-545)
Schach Chapter 11 (pp. 339-356, 369-381)
Schach Chapter 15 (pp. 469-489)

Active Support From Implementation Language

Good code can be written in any language, even assembly language.

BUT, it is much easier and more likely if the language itself actively supports the production of efficient, reliable, maintainable software.

Pragmatic Factors in Language Selection

Dictated by client

Existing expertise/experience

Language used in other projects

Concurrency

Features Language Should/MustSupport/Possess - 1

Modularity The partitioning of a large product into modules

Power and suitability

Simplicity, clarity, orthogonality

Simple, consistent syntax; supportive of clear code includes:

- Method of indicating blocks
- Statements format
- Rules for identifier names

Structured programming

Exception handling

Strong typing

The Case For Strong Typing

"... it is essential that a high-level programming language with strict typing be used for system development. Achieving fault-free software is virtually impossible if low-level programming languages with limited type checking are used."

Sommerville

Features Language
Should/Must Support/Possess - 2

Information hiding

Procedural and data abstraction

Tools

Maintenance

Reuse (libraries, packages)

Support for object-oriented approaches

Mynatt: Procedural and Data Abstraction

To adequately support procedural and data abstraction, a language should include:

Mechanisms for high-level encapsulation of procedural abstractions <u>and</u> data abstractions.

Distinction between the specification of an abstraction and the implementation of an abstraction.

Mechanisms to protect outside access to encapsulated information.

Methods for importing modules from other sources.

L47OH6

Factors Affecting Quality of Source Code - 1

Use of structured coding techniques

Good coding style.

- Shorter is simpler.
- Fewer decisions are simpler.
- Nested logic should be avoided.
- Negative logic should be avoided.

Well-chosen local data structures.

Factors Affecting Quality of Source Code - 2

Well-written internal comments.

- Program headers.
- Internal module headers.
- Line comments.

Readable, consistent source code format and identifier naming.

- Vertical white space.
- Horizontal white space (indentation).
- Readability of comments (spelling, grammar, clarity)

LECTURE NUMBER: 048

TOPIC(S) FOR LECTURE:

Scheduling software projects
Milestones
Work breakdown structures
Network Scheduling Techniques
Introduction to Estimation techniques

INSTRUCTIONAL OBJECTIVE(S):

1. Understand development milestones
2. Recognize the elements of a work breakdown structure
3. Be able to develop a PERT chart
4. Understand slack time and critical paths

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

At this point, you have all participated in all aspects of the software development process and have experienced some of the difficulties with it. You are not alone in this. Show GAO overhead L48OH1. Note that almost 50% of the contracted software was not used. The goals of software engineering can be described as delivering the right product, on-time and within budget. Meeting these goals requires a plan, generally called a software project management plan (SPMP).

(Learning Label- Today we are going to learn ...)

We have provided a basic SPMP for each of your projects. Today we are going to look at how to develop and read SPMPs. We will also see a special notation used for SPMPs.

CONTENTS:

1. There are several unique elements in software that make the development of software projects more difficult than the development of hardware objects. Many authors recognize that there are significant differences between hardware and software that make the development task for each type of product very different. Review the differences mentioned on L48OH2.

   a. Reuse is aided by common interfaces, many of which have been legislated by engineering societies, e.g., the standard electrical plug.

b. The difference in the tangible nature makes it more difficult to track the progress of the development of software. Hardware tends to conform to well know physical laws. Even in those cases where we do not understand the laws, hardware can be experientially tested, e.g. a scientist grows cracks in airplane wings by subjecting them to specific stresses and then recalls those planes which have undergone similar stress.

c. A significant difference between hardware and software is the phase in the development process where errors get introduced. Most hardware errors get introduced in the manufacturing process, while most software errors get introduced in the analysis and design process.

2. Software project management begins with the completion of the requirements. The task needs to be defined before a plan can be made. Discuss the basic steps in the development of a plan. Emphasize that even the SPMP is subject to revision. Requirements are input to the SPMP. L48OH3

L48OH4

3. Divide up the task- people have used the term milestones to indicate major points at which a project should be tracked. We know some major divisions for a software development project. These milestones have been associated with major project deliverables which are clearly defined, e.g., 2167a has a series of clearly defined document deliverables. When dealing with relatively small projects this division of the system is adequate. However when we are talking about programming in the large and large projects such as the environmental control software for the space lab, there is too much time between these milestones to adequately track a project. There are years between these milestones. Another way to handle this problem is to break the major milestones down into smaller, but still discrete and measurable units of work- sometimes referred to as "inch pebbles". The smaller the degree of granularity --the time required to complete one of these task-- the more accurate the scheduling estimates tends to be.

   The milestones and inch pebbles are discrete events that mark the completion of events. These are distinguished from activities that take place over time. Milestones are completions of these project activities. Activities have beginnings and ends and have duration, while milestones are points in time.

L48OH5

4. We can divide the system up into its larger phases and for each phase that takes longer than our desired granularity we can further subdivide the task. The results of this process are called work breakdown structures (WBS) or discrete pieces of work. Within a computer system work breakdown structures are very detailed and include such items as:a description of the task, who is responsible for the task, what resources are needed to accomplish the task, etc.

5. Let us do a preliminary work breakdown list for building a skyscraper. Ask the students to name the tasks in building a skyscraper. List these randomly on the chalkboard. This kind of list does not show interdependence of WBS tasks. Show L48OH6 overhead as your preliminary list and explain the need for each item. Return to this overhead after you have worked through a PERT chart. Ask them to identify which events have to be done in sequence and which events can be done concurrently. One way to display these relationships is to place the WBS activities on an graph called a vertex activity graph- the activities occur at the vertices. This was formalized as a method by the Polaris project and the method is called Performance and Evaluation Review Technique (PERT).

   We can impose an ordering on the WBS list in terms of what things have to be completed before others and what is our estimate of their duration. Show activity list L48OH7. This shows the dependencies for each task in the prerequisite column and the duration of each task in the time column.

L48OH8

6. PERT charts-There are many ways to develop PERT charts. One way is to view an activity as a set of parameters consisting of the WBS #, the earliest possible start date for that activity, the latest possible start date, and the estimated duration of the task. The lines entering a graph node or activity represent those task which must be completed before the activity in that node can be started.

   L48OH9a is a skeleton PERT chart to use as an example. L48OH9b is a completed PERT chart for this example.Begin to develop the PERT chart on the board. Fill in all of the parameters and then point out those places where there is no difference between the earliest possible and the latest start time.

L48OH10

7. Introduce Critical Path Method as an important function of activity networks. Define the critical path and slack time. Discuss how any delay in critical path elements will delay the schedule. Describe how

this process indicates how long it will take to complete a project.

L48OH11 Summarize the definitions and discuss this overhead.

8. False safety factor- Sometimes we think that the way to meet the schedule determined by a PERT chart is to just add some time. For example, Sommerville pg 503 "increase the estimate to cover anticipated problems and add a contingency factor to cover unanticipated problems." However studies have shown that this is not an effective technique. Adding time to a schedule actually makes a project take longer and the same unexpected surprises occur later in the project.

9. Once a PERT chart and its accompanying schedule are done. The schedule should immediately be reviewed for those typical items that impact a schedule but are often forgotten. How will the vacation schedule impact the availability of personnel? How long will it take to train the staff on the new system or CASE tools? What other projects are planned such as migrating to a new operating system in the middle of your project? These will all negatively impact the project schedule. What is the budget cycle of the external agency. Many of those government projects which were not delivered were funded in September but were not re-funded in October, the beginning of the federal governments fiscal year.

10. Discuss the two standards for SPMPs on L48HD1 and L48HD2.

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

<u>vocabulary introduced</u>:

activity networks
CPM
risk analysis
PERT
slack time
critical paths
milestones
work breakdown structures

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>:

L48OH1	GAO survey of software contracted for by government
L48OH2	Hardware vs. software development

L48OH3	The software project management life cycle
L48OH4	Milestones vs. inch pebbles
L48OH5	Work breakdown structure
L48OH6	Planning Exercise
L48OH7	Activity dependencies - tasks, time, and prerequisites
L48OH8	Contents of the graph node
L48OH9a	Pert chart Outline
L48OH9b	Pert chart Completed
L48OH10	Critical path method
L48OH11	Critical path definition

handouts:

L48HD1	IEEE project plan outline
L48HD2	NASA project plan

RELATED LEARNING ACTIVITIES:
(labs and exercises)

READING ASSIGNMENTS:
Sommerville Chapter 25 (pp. 477-492)
Sommerville Chapter 26 (pp. 495-507)

RELATED READINGS:
Ghezzi Chapter 8 (pp. 415-440)

GAO Survey of Software Procured by Government

Category	Amount (millions)	Percent
Used as delivered	$0.119	1.75%
Used after changes	$0.198	2.91%
Used but reworked or later abandoned	$1.3	19.12%
Paid for but not delivered	$1.95	28.68%
Delivered but NEVER successfully used	$3.2	47.05%
Total cost of software surveyed	$6.8	100%

Hardware versus Software Development

Reusable Parts

Tangible

- Physical laws
- Experiential standards

Source of problems

- In development
- In maintenance

Testing Standards

The Software Project Management Life Cycle

I. Prepare the SPMP
   1. Examine the functional and non-functional requirements
   2. Divide the project up into understandable units and required deliverables
   3. Review the items in 2. and add to them the derived deliverables
   4. Build work breakdown structures for each major task
   5. Develop an activity network for the project schedule.
   6. Develop an SPMP

II Execute the SPMP
   1. Start activities according to the schedule
   2. Frequently monitor the project schedule
   3. Modify the SPMP or internal subplans as needed.

Milestones Versus Inch Pebbles

2167a

Requirements Specification Review

Preliminary Design Review

Detailed Design Review

Code and Test

System Test

Acceptance Test

Inch Pebbles

Activities versus Events

Work Breakdown Structure

Task name or ID: (critical path ?)
Description of task:
Dependencies:(Predecessors)
Project members:(skills needed)

Duration: Start Date: End Date:

Resources Required:
Entry Criteria:
Completion Criteria:
Responsible Staff member:

Acceptance Criteria:
Exit Criteria:

Sign-off person:

Task deliverables:

validation complete	date__
documents complete	date__
reviewed	date__
CMS	date__

Risks:

Risk Plan:

Planning Exercise

Project: Build a skyscraper:

Tasks: (WBSs)

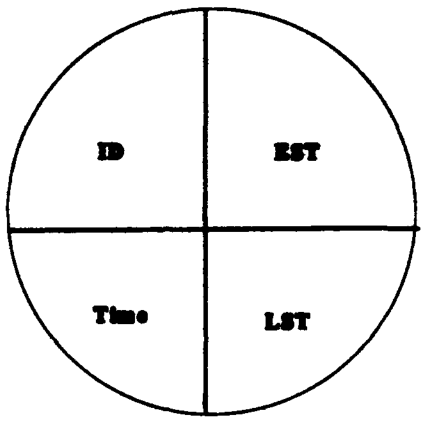
Fence off site
Erect Workshops
Dig foundation
Install on-site concrete plant
Bend reinforcing rod
Fabricate steel work
Paint steel work
Place reinforcements
Pour foundation
Erect steel work
Place a tree

Activity Dependencies

Project Build a building:

	Tasks	Time	Prereq.
1	Fence off site	2	0
2	Erect Workshops	4	1
3	Dig foundation	6	1
4	Install on-site concrete plant	4	1
5	Bend reinforcing rod	3	2
6	Fabricate steel work	7	2
7	Paint steel work	3	6
8	Place reinforcements	5	3,5
9	Pour foundation	8	4,8
10	Erect steel work	10	9,7
11	Place a tree	0	10

Contents of the Graph node



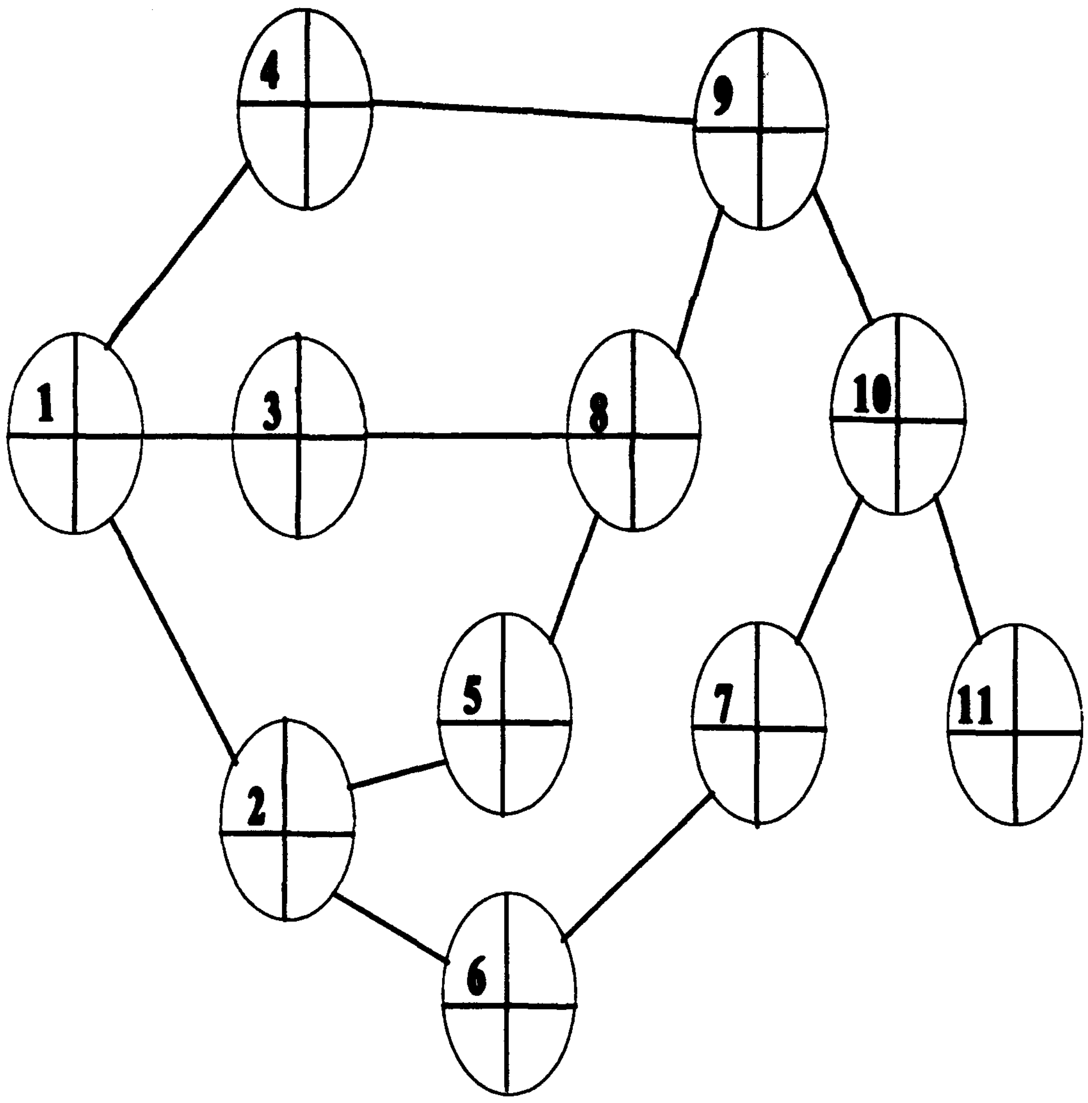
<u>ID</u> = WBS task id number

<u>EST</u> (earliest start time) = Max{pred(TIME) + pred(EST)}

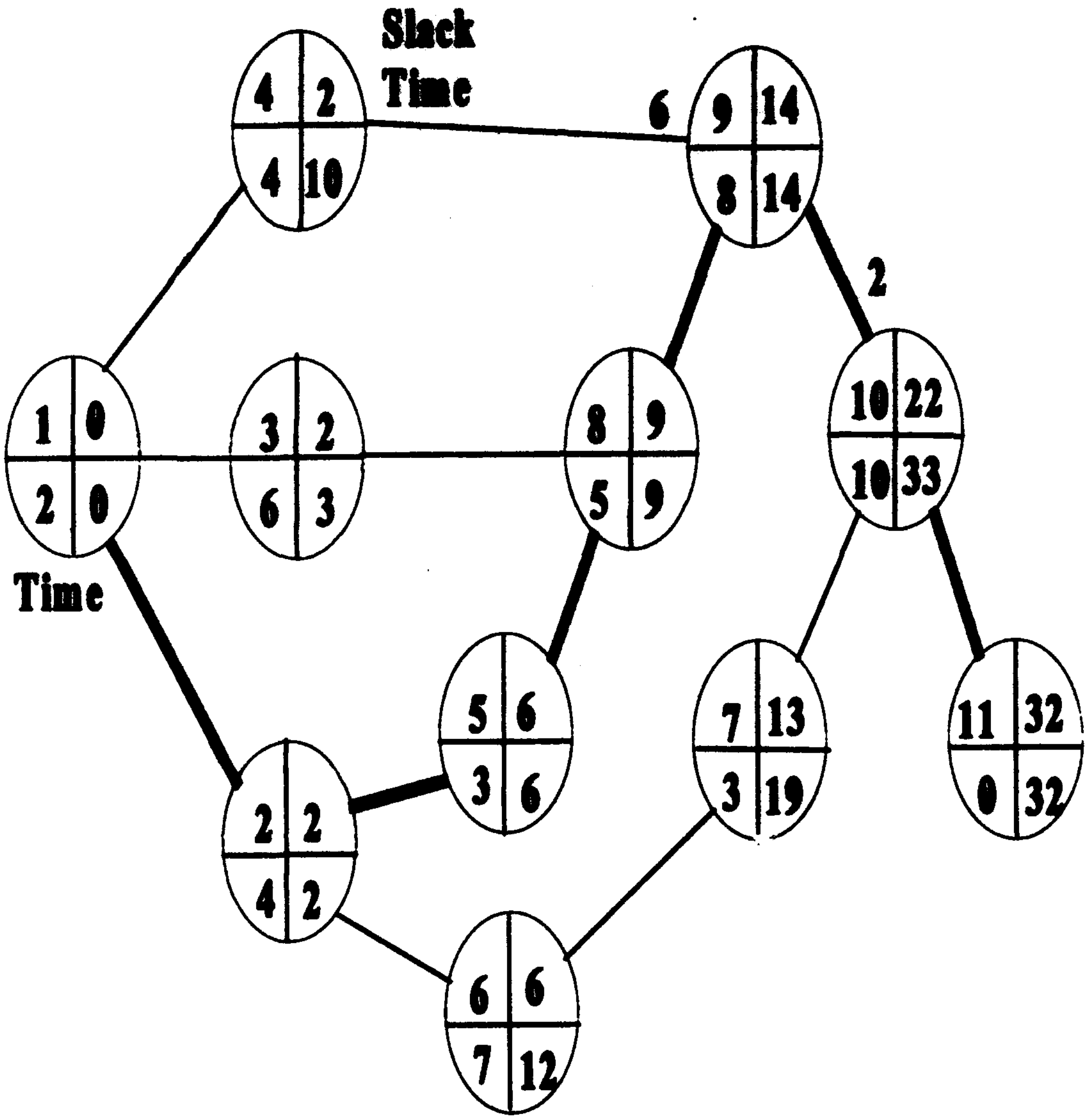
<u>Time</u> = anticipated duration of this activity

<u>LST</u> (latest start time) = Min {succ(LST)- TIME}

Example Pert Chart Outline



Example Pert Chart Completed



Critical Path Method (CPM)

Slack time = LST - EST;

Critical path = path whose slack time = 0.

Time to complete project

= sum of (max of (EST (PRED Time) + TIME)

Critical Path Definition

The critical path is the longest path through the network in terms of the total duration of tasks

In complicated projects many "near critical" tasks and paths may exist

Delays in a non-critical path task may result in a new critical path

Lengthening the critical path lengthens the project

Questions Answered by Critical Path Analysis

What is the minimum elapsed time to complete the project?

What tasks determine whether the project is completed in the minimum time?

What is the latest time we can start a particular activity without impacting the overall finish time?

IEEE Project Plan Outline (1):

Title Page
Revision Sheet
Preface
Table of Contents
List of Figures
List of Tables

1. Introduction
    1.1 Project Overview
    1.2 Project Deliverables
    1.3 Evolution of the SPMP
    1.4 Reference materials
    1.5 Definitions and Acronyms

2. Project Organization
    2.1 Process Model
    2.2 Organizational Structure
    2.3 Organizational Interfaces
    2.4 Project Responsibilities

IEEE Project Plan Outline (2):

3. Managerial Process
3.1 Management Objectives and Priorities
3.2 Assumptions, Dependencies, and Constraints
3.3 Risk Management
3.4 Monitoring and Controlling Mechanisms
3.5 Staffing Plan

4. Technical Process
4.1 Methods, Tools, and Techniques
4.2 Software Documentation
4.3 Project Support Functions

5. Work Elements, Schedule, and Budget
5.1 Work Packages
5.2 Dependencies
5.3 Resource Requirements
5.4 Budget and Resource Allocation
5.5 Schedule

Additional Components
Index (optional)
Appendices (optional)

NASA-Sfw-DID-Ada (1):

1.0 Introduction
- 1.1 Identification
- 1.2 Scope
- 1.3 Purpose
- 1.4 Organization
- 1.5 Objectives
- 1.6 Program Constraints
- 1.7 Program Software Schedules
- 1.8 Program Controls

2.0 Applicable Documents
- 2.1 Reference Documents
- 2.2 Information Documents
- 2.3 Parent Documentation

3.0 Resources & Organization
- 3.1 Project Resources
  - 3.1.1 Contractor Facilities
  - 3.1.2 Government Furnished Equipment, Software & Services
  - 3.1.3 Personnel
- 3.2 Responsibilities
- 3.3 Panels
  - 3.3.1 Review Panels
  - 3.3.2 Advisory Panels
- 3.4 Software Development
  - 3.4.1 Organizational Structure - Software Development

NASA-Sfw-DID-Ada (2):

3.4.2 Personnel - Software Development
3.4.3 Resources - Software Development
3.4.4 Software Development Tools, Techniques, Methodologies
3.4.5 Software Environment Section

4.0 Life-Cycle Management
- 4.1 Concept and Project Definition
- 4.2 Software initiation
- 4.3 Software Requirements Definition
- 4.4 Software Preliminary Design
- 4.5 Software Detailed Design
- 4.6 Software Implementation
- 4.7 Software and System Integration and Testing
- 4.8 Software Acceptance Testing
- 4.9 Sustaining Engineering

5.0 Management Controls

- 5.1 Engineering Master Schedules and Risk Management
  - 5.1.1 Activities
  - 5.1.2 Activity Network
- 5.2 Engineering Master Schedule Reviews and Reporting Policies
- 5.3 Risk Management
  - 5.3.1 High Risk Areas
  - 5.3.2 Technology Risks
  - 5.3.3 Disaster Risks and Recovery
- 5.4 Status and Problem Reports

NASA-Sfw-DID-Ada (3):

6.0 Software Support Environment
- 6.1 Software Development
- 6.2 Software Acquisition
- 6.3 Software Integration
- 6.4 Operation and Maintenance
- 6.5 Software Tools

7.0 Software Product Assurance
- 7.1 Software Configuration Management
- 7.2 Software Independent Verification and Validation
- 7.3 Software Security
- 7.4 Software Product Assurance
- 7.5 Software Interface Definition and Control
- 7.6 Waivers to SPMP Policies and Procedures
  - 7.6.1 Permanent Waivers
  - 7.6.2 Temporary Waivers
  - 7.6.3 Tool and Testbed Waivers

8.0 Notes
9.0 Appendices
10.0 Glossary

LECTURE NUMBER:049

TOPIC(S) FOR LECTURE:

COCOMO
Lines of code estimation techniques
Function Points and lines of code

INSTRUCTIONAL OBJECTIVE(S):

1. Use COCOMO equations
2. Do a function point analysis
3. Develop other function point metrics

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

We have seen the difficulties that arises in trying to correctly schedule a software project and the allocation of resources needed for its development. These resources have to be paid for. Most of us want to know how much something is going to cost before we order it. Software is no different. We would like to know how much we are going to pay for software before we purchase it. This is no problem for packaged software; we simply look at the price on the box, but how do we determine the cost of software which has not yet been developed or even been thought of before. What do you say when the president of the United States tells you she/he wants a system that will let everyone vote in federal elections from their homes and asks you how much will it cost to develop a "Home Voting and tabulating system"? How can you determine what it would cost to develop the project you have just completed?

(Learning Label- Today we are going to learn ...)

There are reasonable ways to approach this problem of estimating the cost of developing software. Today we shall look at two of them - The Constructive Cost Model (COCOMO) and Function Point Analysis.

CONTENTS:

1. COCOMO is a series of formulae which, using an estimate of the total number of lines needed for the product, produces: cost estimates, scheduling estimates and manpower requirements for each phase of the life cycle.

L49OH1

2. Describe the origin of COCOMO in a business environment and describe how it was empirically derived. This is important to follow up

on so that the students understand that COCOMO and most other estimation techniques can be empirically calibrated for better accuracy. Although it does not presuppose a perfect environment, COCOMO makes several assumptions about the development environment of the project being estimated. Review these presumptions.

3. COCOMO does not make the mistake of presuming that all estimates start from the same point. Sometimes a rather quick estimate is needed based on relatively little detail. This is a Basic estimate. Discuss the intermediate level of detail used for an estimate and use this as an opportunity to discuss the different project cost drivers used for an intermediate estimate. The students find it easy to understand how slow turn around time and high reliability requirements impact software development time. Time discussing the cost drivers is well-spent.

4. After the discussion of cost drivers it is easy for the students to understand the impact of the different development environments. These environments represent a scale of difficulty. The software development process is more stable and easier to understand and control in an organic environment.

   a. The organic environment is characterized by a thorough understanding of the product and product objectives, extensive experience with related systems, limited pre-defined requirements, limited commitment to externally defined interfaces, minimal need for innovation and a low need for an early completion date.

   b. You should plead ignorance about the origin of the label "semi-detached". The semi-detached development environment is between the organic and the embedded environment. The semi-detached environment requires considerable : understanding of project objectives, experience with related systems, need to meet predefined requirements, and need for conformance to external interfaces. It also involves some concurrently developed hardware and some need for innovation.

   c. Embedded development must meet predefined requirements conform to external interfaces and requires concurrently developed hardware. It also requires considerable innovation and has a high need for early completion.

L49OH2

5. Discuss COCOMO as almost a single driver estimation tool which provides equations for each development environment and the only variable is delivered source instructions(DSI). Put off any discussion for awhile about how you derive the number of DSI. Work through each of the person-month equations. Carefully distinguish between the person months (or effort required) and the time to develop(TDEV) the software or the period of time in which that effort can be squeezed. Go through the TDEV equations.

6. Other results- show that it is relatively easy to determine the average number of personnel needed during the life of the project -PM/TDEV- but that this does not tell you how many people you need during any particular phase. Introduce the Rayleigh Curve here as a mean of determining how many people you need for each phase of the life cycle.

7. An example of how to calibrate COCOMO is given in the Ada modifications to COCOMO which were also done by Barry Boehm. L49OH3

8. a major criticism of COCOMO is that it is difficult to initially estimate line of code before a system is actually written. Function points is one way to determine DSI. When you introduce function points (FP) be sure to emphasize that FP is a dimensionless number. It does not really count "FUNCTIONS'.
   a. Go over the general FP equation. Then discuss how they derive the sum of functional factors.

   L49OH5
   b. Show the chart indicating how to adjust the complexity of the function points and derive a total L49OH4. This provides some discrimination among type of inputs and outputs which is based on complexity.

   L49OH6
   c. Then show the chart o· degrees of influence and weighting factors L49OH7.

   d. Work through a set of numbers and arrive at a FP. Ask the student what this number represents. They will have trouble coming up with an answer. Make clear that this is a number that can be used to compare software projects if the same mechanism used on both projects to derive function points.

L49OH8

8. Uses of Function points- They can be used as a productivity measure or a quality test . We started to look at function points as a way to determine the lines of code that a system might have before the code was even written. Using past experience we have determined that the lines of code that it takes per function point varies from language to language. A correlation between function points and lines of code, for one development environment, is shown. If it is estimated that a project will take 200 function points and it is being written in Ada then it will take approximately 14400 lines of code. This lines of code approximation can be used in the COCOMO equations.

9. Emphasize that these numbers are approximations and are different for different development environments. But these values can be calibrated for any consistent development environment.

PROCEDURE:

teaching method and media:

vocabulary introduced:

COCOMO
Person-month equations
Time of development equations
Function points
Weighting factors
Degrees of influence

INSTRUCTIONAL MATERIALS:

overheads:

L49OH1	COCOMO
L49OH2	COCOMO equations
L49OH3	Ada modifications to basic COCOMO
L49OH4	How to determine code size
L49OH5	Functional point counting
L49OH6	Degrees of influence
L49OH7	Weighting factors
L49OH8	Additional function point metrics

handouts:

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 037 - Function Points

READING ASSIGNMENTS:

Sommerville Chapter 27 (pp. 511-533)
Mynatt Chapter 1 (pp. 17-27)

RELATED READINGS:

Ghezzi Chapter 8 (pp. 415-433)
Pressman Chapter 3 (pp. 83-89)
Schach Chapter 8 (pp. 212-215)

COCOMO

Assumptions of the Basic COCOMO Model:

- Requirements
- Low volatility
- Good S.E. practice
- Good management

Three levels of detail:

- Basic -
- Intermediate -
- Detailed -

Three development environments:

- Organic -
- Semi-detached -
- Embedded -

DSI:

- Delivered
- Source
- Instructions

COCOMO Equations

COCOMO Person-month Equations:

Organic: $PM = 2.4 * KDSI^{1.05}$

Semi-detached: $PM = 3.0 * KDSI^{1.12}$

Embedded: $PM = 3.6 * KDSI^{1.20}$

COCOMO Time of Development Equations:

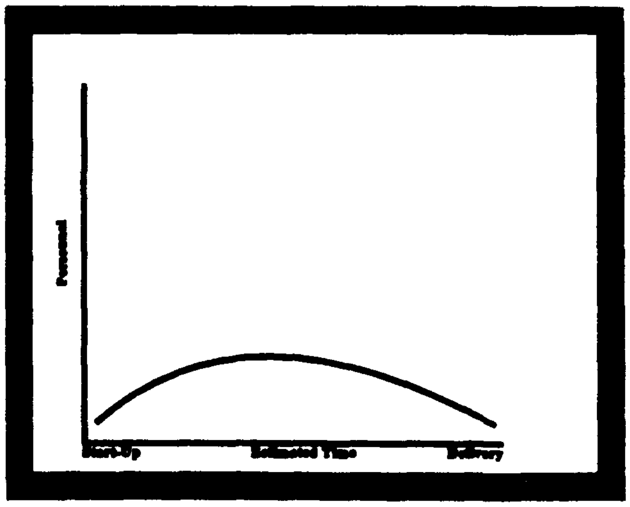
Organic: $TDEV = 2.5 * (PM)^{0.38}$

Semi-detached:$TDEV = 2.5 * (PM)^{0.35}$

Embedded: $TDEV = 2.5 * (PM)0.32$

Full-time Software Personnel = PM/TDEV

Work distribution:



Ada Modifications to Basic COCOMO

Assumes smaller teams initially

Longer design period

$PM = 2.8 * KDSI^{1.04}$

$TDEV = 3 * PM^{0.32}$

PM for PD, DD, CUT, IT =
PM * (.23, .29, .22, .26)

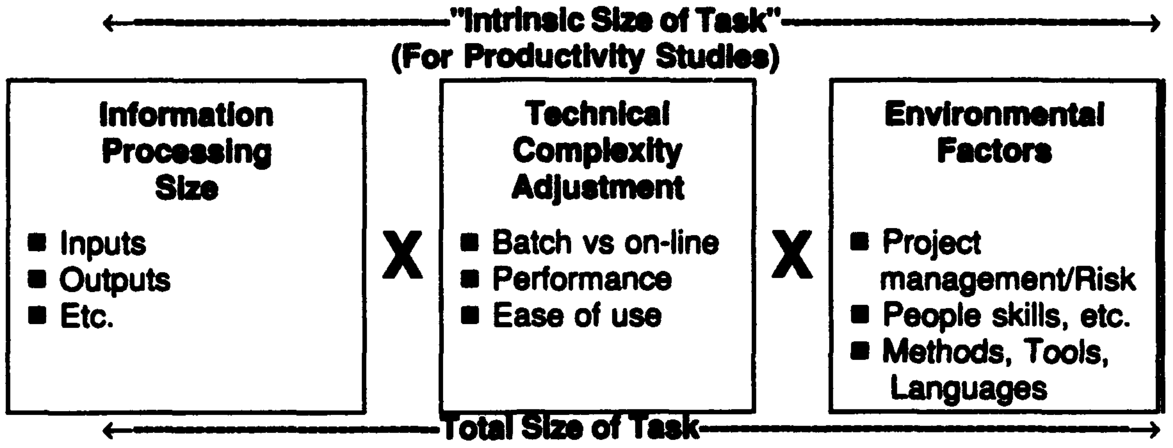
TDEV for PD, DD, CUT, IT =
TDEV * (.39, .25, .15, .21)

How To Determine Code Size

Function Points =

Sum of Functional Factors *
[0.65 = 0.01 * Sum of Weighting Factors]

Five Functional Factors
- Number of user inputs
- Number of user outputs
- Number of user inquiries
- Number of files
- Number of external interfaces



Unadjusted Functional Point Counting

	Level of Information Processing Function			
Description	Simple	Average	Complex	Total
External input	x 3 =	x 4 =	x 6 =	
External output	x 4 =	x 5 =	x 7 =	
Logical internal file	x 7 =	x 10 =	x 15 =	
Ext. interface file	x 5 =	x 7 =	x 10 =	
External Inquiry	x 3 =	x 4 =	x 6 =	
			Total unadjusted function points	

Technical Complexity Factor

ID	Characteristic	DI	ID	Characteristic	DI
C1	Data communications		C8	On-line update	
C2	Distributed functions		C9	Complex processing	
C3	Performance		C10	Re-usability	
C4	Heavily used configuration		C11	Installation ease	
C5	Transaction rate		C12	Operational ease	
C6	On-line data entry		C13	Multiple sites	
C7	End user efficiency		C14	Facilitate change	
				Total degree of influence	

TCF = 0.65 +0.01 x (Total Degree of Influence)

		DI Values
TCF -	Technical complexity factor	
DI -	Degree of influence	Not present or no influence = 0
FP -	Function points	Insignificant influence = 1
UFP -	Unadjusted function points	Moderate influence = 2
		Average influence = 3
		Significant influence = 4
		Strong influence, throughout = 5

Thus each degree of influence is worth 1 percent of a TCF which can range from 0.65 to 1.35.

The intrinsic relative system size in Function Points is computed from

FP's = UFP's X TCF

Function points are therefore dimensionless numbers on an arbitrary scale.

Weighting Factors

Does the system require reliable backup and recovery?

Are data communications required?

Are there distributed processing functions?

Is performance critical?

Will the system run in an existing, heavily utilized operational environment?

Does the system require on-line data entry?

Does the on-line data entry require the input transaction to be built over multiple screens or operations?

Are conversion and installation included in the design?

Is the system designed for multiple installations in different organizations?

Is the application designed to facilitate change and ease of use by the user?

Additional Function Point Metrics:

Productivity = Function Points/ Person-Months

Quality = Errors/Function Point

Cost = $/Function points

Documentation = Pages/Function Point

Correlations between function points and lines of code:

Basic assembler	360
Macro assembler	213
C	128
COBOL	105
FORTRAN	105
Pascal	80
Ada	72
Basic	64
4GL	25

LECTURE NUMBER:050

TOPIC(S) FOR LECTURE:
Assessment of projects

INSTRUCTIONAL OBJECTIVE(S):
(indicate learner behavior expected or learning outcome)

1. Encourage students to compare different development methodologies
2. Encourage students to compare different development environments
3. Review significance of each phase of the development life-cycle

SET UP, WARM-UP:
(*How involve learner*: recall, review, relate)

We have completed work on several different types of projects, using different methodologies and team organizations. Each of these differences impact not only the final product developed but also the process of development.

(Learning Label- Today we are going to learn ...)

Now that you have completed the project work for this course. We should take some time to re-examine some of the processes we have been through.

CONTENTS:

1. Introduce a series of discussions on the various methodologies. Project 1 employed a structured methodology, while project 2 used an object-oriented methodology. Would an object-oriented(O-O) methodology have been better for project 1? Try to direct the discussion away from the questions of how hard it would be to learn object-oriented methodologies quickly and direct the discussion to the appropriateness of the O-O methodology. Get them to articulate the difference between the methodologies.

2. The projects differed in terms of structure and the communications required. Discuss whether the structure imposed on the extended project should have been used on the small project as well. This opens a good discussion on the positive and negative impacts of controlling disciplines- CM, SQA, V & V. At some point they will start to talk about the problems of team interactions. Direct this to an issue of the importance of good team communication and away from question of conflicting personalities.

3. Questions of effective intra-team communications can be discussed here along with the importance of clear and baselined documents.

This can be used to review each stage of the life cycle.

4. An assessment which we used in a previous class is included as instructor notes. The questions we asked can be used to develop you own assessment questions.

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

<u>vocabulary introduced</u>:

<u>INSTRUCTIONAL MATERIALS</u>:

<u>overheads</u>:

<u>handouts</u>:

<u>RELATED LEARNING ACTIVITIES</u>:
(labs and exercises)

<u>READING ASSIGNMENTS</u>:
None

<u>RELATED READINGS</u>:
None

Software Engineering Survey

In an effort to improve the structure of this course and the quality of its instruction we ask that you complete this survey. We value your judgements. Your thoughtful responses will *significantly* add to the quality of this course.

This questionnaire is a significant portion of the final examination. You will get FULL credit for this question simply by giving us your thoughtful responses.

A PROJECT 1 (Small project names)

The purpose of project 1 was to give you a rapid introduction to software engineering and to let you quickly experience the entire software lifecycle by taking a small project from requirements through implementation.

1. Were these projects appropriate for this purpose?

2. The team organization used for this project is called a democratic team. Did this organization help or hinder your progress on the project.

3. Would it have helped to have one or more of the teachers present at some of your team meetings?

4. How did you keep track of what tasks had to be done and whether they were done?

B PROJECT 2 (Extended project name)

In this project we used a "matrix model" of team organization.

1. Did the experience of working on several different teams during the project help you better appreciate the additional problems in developing more complex software? Did you find it a helpful experience to work on several teams? (In the matrix structure you switched teams several times. How did you feel about that?)

2. Unlike project one, here you had to interact with other teams. What did you think of this experience?

3. One of the modifications we are considering for future use is the establishment of a "tools team" that would learn all the CASE software tools and provide training and on-going help in the use of the tools. What do you think about this idea?

4. Another modification we are considering for future use is the establishment of a "user interface team" that would be responsible for the user manual and the design of the user interface. What do you think about this idea?

5. Can you think of any other teams that might be helpful?

6. In this project we introduced configuration management. In what ways did this help your work on the project? How might it be improved?

7. How did the use of object-oriented design help on this project?

8. There were two different software project management plans provided to you for this project. Was the more detailed plan (inch pebbles) more help to you?

9. Some people say it is very hard to shift from a structured analysis model (DFDs, Context Diagrams, data dictionary, process specifications) to an object-oriented model.

   a. How did the requirements list and the structured analysis model help you to make the transition?

   b. How did our discussion of Ada and our explanations of the object-oriented model help you make this transition?

C REVIEWS

1. We required a number of reviews (requirements, preliminary design, test plan...) as the projects were developing.

   a. List what you consider the major strengths of the in-class reviews.

   b. List what you consider the major weaknesses of the in-class reviews.

   c. How could we respond to these weaknesses?

2. We required that each team member take part in these reviews. How did this help you?

3. We are thinking of assigning some students who were not part of the team as reviewers for each presentation. How do you think this might improve the review process?

D TEAMS

1. In project one we used a democratic team, in project two a matrix organization, and in the maintenance exercises a chief-coordinator organization. Which structure did you find the most productive? Which structure did you find the most comfortable?

2. We assigned students to teams based in part on student input. How do you think we should assign students to teams?

E Ada

1. Was it helpful to implement the large project in a new language? (Try to separate your answer from the question of the adequacy of our current Ada environment)

2. Did the use of Ada help you see the application of software engineering principles in specifications? In design?

3. Ada was gradually integrated into this course; first by showing specification examples and then by showing program segments. We read through code in class. We revisited Ada code several times in increasing depth. Was this technique (called program reading) a helpful way to introduce Ada as opposed to simply studying syntax?

F EVALUATION

1. Please comment on the peer evaluation process used in assessing the projects.

2. Were the examinations fair and did they adequately cover the material? How would you like to change the structure and content of these tests.

3. Can you think of any other ways we might use to improve our evaluation of your work in the course?

4. We gave a variety of feedback as the projects progressed: first draft reviews, review evaluations, meetings with customers or users etc. Was this adequate and helpful? Were there some other things we could have done to help your team's progress.

5. Were the software project management plans for the projects adequate?

G OVERALL COURSE APPROACH

1. This course involved three professors. How was this helpful and how was it not helpful? Would you like to see other classes "team-taught"?

2. One of the major goals of this course was to give you experience in a real world software development environment, functioning in a variety of roles and working with a variety of people.

   a. Do you think that the course adequately reflected the real world?
   b.. Do you feel that this course helped prepare you for working in an industrial environment?
   c. How would you modify the types and the number of software projects to help accomplish this goal.

3. In our spiral" approach we briefly introduced a subject in one lecture, then in later lectures we provided additional detail. In some cases subjects were revisited several times, each in increasing depth, e.g., testing, design, or configuration management. Do you think this approach helped in building your understanding?

4. The spiral approach was also used in the projects. Did this approach help? For example, would the extended project have been harder for you to do if you had not first done a smaller project?

5. Was the coordination of the lecture, projects, and readings effective?

6. List what you consider the major strengths of this course

7. List what you consider the major weaknesses of this course
8. How might we still meet our goals and respond to these weaknesses?

H TEXTBOOKS

1. Comment on the value of the Sommerville textbook to the course.

2. Comment on the value of the Mynatt textbook to the course.

3. Comment on the value of the Benjamin textbook to the course.

If there is anything else you want to comment on, then please do?

Real-World Software Engineering

III LABS

a. Laboratory Meetings.

Structured labs are utilized throughout the course. By structured labs we mean meetings held at the same time as regular class meetings and during which the instructor was present providing guidance and support for the students' efforts, particularly team efforts.

Laboratories are utilized for a variety of activities, most of which are directly related to the team projects. They provide opportunities for student teams to meet, for instructor interaction with teams, and for customer or user interaction with teams. Lab activities included organizational meetings, customer requests, distribution and discussion of project materials (e.g. software project management plan, meeting report templates, presentation guidelines and assessment forms), structured exercises on methods and tools, feedback to teams on deliverables and presentations, formal reviews, code inspections, and "steering" of teams where necessary.

The amount of faculty direction varies from lab to lab. Some labs consist of exercises where the student interacts directly with the instructor. Some labs are used to demonstrate a CASE tool that is being used for a segment of a project. Other labs consist primarily of team meetings. Less structured labs are useful when deadlines for major deliverables approached. These labs are primarily dedicated to team meetings to work on team deliverables. The connection of the lab time to the projects is further reinforced by having reviews of team deliverables during lab time.

We believe it is essential to have student labs scheduled at the regular class meeting times. Students have no conflicting obligations at that time and any scheduling difficulties for team meetings are eliminated. Equally important is the availability of the instructor. The instructor is an immediate resource to provide whatever level of steering is necessary to keep the team effort on track. While the instructor should avoid micro-management of team projects, he/she must be willing to intervene at times. Availability of the instructor has other advantages as well. Teams are sometimes composed of personalities that do not easily co-exist. The students' knowledge that the instructor is close by helps to minimize the eruption of overt hostilities. Your availability to the students reinforces the notion that you _and_ they are working toward a single goal -- the development of a successful software project.

Be flexible with lab time. Use it to keep the projects on-track. Always provide some guidance for the lab. You must do what you can to avoid the tendency some students have to think that lab time is just an opportunity to leave early. Your presence in the lab or being readily available during labs will effectively minimize this attitude.

b. Introduction to lab forms

The lab form is similar to the lecture form and provides a consistent structure for each of the labs. Each lab description consists of the following sections.

<u>LAB NUMBER</u> - The labs are numbered sequentially and the number is used to cross reference lectures and laboratories.

<u>TOPIC(S) FOR LAB</u> - These usually refer to the projects and indicate the specific activity and/or project deliverable to which the lab is related.

<u>INSTRUCTIONAL OBJECTIVE(S)</u> - Again, these usually refer to the projects and are generally stated in terms of behavioral goals.

<u>ASSOCIATED LECTURE NUMBER</u> - This identifies the particular lecture with which this lab is associated. In many cases these labs relate the lecture material to the practical concerns of their projects.

<u>PROCEDURE</u> - A step-by-step description of the activities to be conducted is provided. Handouts or overheads are referenced where used.

<u>ASSOCIATED HANDOUTS</u> - A list of any handouts used during the lab is provided. A copy of each handout for a lab immediately follows the lab form.

c. Labs

The lab forms for the individual laboratories follow.

LAB NUMBER: 001

TOPIC(S) FOR LAB:

Customer requests
Small projects team organization
Requirements

INSTRUCTIONAL OBJECTIVE(S):

1. Understand specific customer request.
2. Meet fellow team members for small project.
3. Develop an abstract and list of requirements for the small team project.

ASSOCIATED LECTURE NUMBER:

Lecture 003

SET UP, WARM-UP:

Recall in the preceding lectures, you were introduced to the process of developing a requirements for a requested system and typical problems related to customer requests. We characterized the process of developing good requirements as one of extraction. We also talked about the importance of understanding the system and developing and specifying the requirements. Today you're going to become a member of a project team, listen to a customer who wants your team to develop a system, and begin work on developing the system requirements.

PROCEDURE:

1. HANDOUT(S) - Project descriptions.
The instructor(s), role-playing as customer(s), present their requests to the class. The request is in the form of a written narrative description and an oral description. Any questions are discussed and answered from a customer perspective. It is important to resist the tendency to deal, as the customer, with technical questions; the customer has domain expertise but not software engineering expertise. Students have not yet been assigned to teams or projects. All students listen to all customer requests at this point. [Note -- a paper entitled "Bringing the Customer into the Classroom" is included in the projects section of this packet.]

2. HANDOUT - Software project management plan (SPMP)
Discuss the plan with particular attention to configuration items (CIs), presentations, and schedule.

3. HANDOUT - Small project team and project assignments
Announce and distribute the project team and project assignments. Note

that no organizational structure is imposed; a democratic team organization is implied. Teams are encouraged, as a first order of business, to determine some times when they can all meet. Have the team fill out a contact sheet containing the name and phone number of each team member.

4. HANDOUT - Team meeting report template
Inform the teams that a record is to be submitted for each team meeting. The form provided is to be used. At the first class meeting of each week teams are to submit their meeting records for each meeting from the previous week. Make the templates available in electronic form.

5. Teams are given the remainder of the period to meet and begin development of a list of requirements (CI-1). Their customer is available for the remainder of the class period to answer questions regarding the requested system.

   HANDOUT - disk with client request
   Each team is provided with a disk containing the preliminary client request. This will be refined for the first configuration item (see SPMP).

ASSOCIATED HANDOUTS:
1. Project description(s) - preliminary client request
2. Software Project Management Plan (SPMP)
3. Team and project assignments
4. Disk containing preliminary client request
5. Template for team meeting records

PROJECT DESCRIPTION: KIOSK VENDING MACHINE

A system is needed to control a kiosk vending machine that consists of three apparently separate vending machines that are actually under common control. The kiosk has three walls, each wall housing one vending machine. Each machine can dispense up to 32 different items and has its own coin slot, dollar bill slot, and selection panel. The coin slots accept quarters, dimes and nickels. The dollar bill slots accepts only one-dollar bills. The selection panels consist of a series of buttons, each showing a graphical representation of the item to which the button corresponds or an "empty" indicator.

To use any of the machines, a customer enters money, presses one or more buttons on the selection panel, and then presses a "dispense" button. Assuming sufficient money has been entered, the selected items are dispensed and the correct change returned. A customer can cancel a transaction at any time prior to hitting the "dispense" button and his/her money is returned. If a customer's requests cannot be honored, his/her money is also returned automatically.

All three machines are to have a common control system that keeps track of each machine's status including the total amount of money it has taken in, and the number of items dispensed (for each of the 32 different items). The money supply and money input is shared by the three machines and the system must keep track of the number of coins it has (quarters, dimes, and nickels), and number of dollar bills it has.

A maintenance operator services the kiosk frequently. The operator must be able to request a report of the kiosk status as well as the status of any of the individual machines. The operator must also be able to restock the machines, reprice items, and replenish and collect money.

There are several mechanical functions that can go awry and the existence of these problems are indicated by an alarm which is transmitted to the operator's pager. Alarm conditions always indicate the machine involved and the particular condition. Conditions include a stuck item, stuck coin or dollar bill slot, machine low on money or type of change, machine out of money, machine out of particular items, and machine or kiosk door open. The operator needs the ability to turn the alarm indicators off. Stuck items or coins disable the particular machine until it is serviced. The machine out of money condition disables the kiosk until it is serviced. A problem analysis report is generated monthly.

PROJECT DESCRIPTION: GAZEBO LOTTERY SYSTEM

A small town has decided to operate a local lottery. A software system is needed to control it. Housed in a gazebo in the center of the town square, it consists of three apparently separate lottery ticket machines that are actually under common control. A structure inside the gazebo has three walls, each housing one lottery ticket machine. Each machine can dispense up to three different types of lottery tickets and has its own coin slot (quarters only), dollar bill slot (ones only), and selection panel. The selection buttons each show a graphical representation of the type of lottery ticket to which the button corresponds, or, an "empty" indicator.

To use any of the ticket machines, a customer enters money and then uses the selection panel to select the type of lottery tickets and the numbers to play. The customer may make multiple selections and, for each, he/she enters the ticket type (daily, weekly, or monthly) and five numbers (numbers from 1 to 20 can be selected). After all selections, the customer presses a "dispense" button. If sufficient money has been entered (initially all tickets are $1), the selected lottery tickets and correct change are dispensed. A transaction can be cancelled at any time prior to hitting the "dispense" button and the money is returned. If a customer's selections cannot be honored for any reason, his/her money is also returned. Each lottery ticket dispensed contains the time and date, the machine number, and the five numbers selected.

All three lottery ticket machines share a common control system that keeps track of each machine's status including the total amount of money taken in, the number of lottery tickets dispensed (for each of the three different types), and the numbers selected. The money supply and money input is shared by the three machines and the system must keep track of the number of coins and dollar bills.

An operator is present whenever the gazebo is open, and performs system start-up and shut-down at the beginning and end of each day, respectively. The operator can request a report of the gazebo status as well as the status of any of the individual lottery ticket machines. The operator must be able to restock the ticket supply, reprice tickets, and replenish and collect money. At the end of each day, the operator enters the winning numbers for the daily lottery and requests a daily lottery report including the date, winning numbers, number of winners, and the amount due each winner. Seventy-five percent of the money is distributed evenly among the winners; twenty-five percent goes to the town. In a similar fashion, once a week the operator requests the weekly lottery report, and once a month he/she requests the monthly lottery report.

Several mechanical malfunctions can occur and their existence is indicated by an alarm which is monitored by the operator. Alarm conditions indicate the ticket machine involved and the particular condition. Conditions include stuck ticket dispenser, stuck coin or dollar bill slot, machine low on money or type of change, machine out of money, machine out of tickets, and machine or door open. The operator must be able to turn the alarm indicators off. Stuck ticket dispensers or coins disable the particular machine until it is serviced. The machine out of money condition disables the gazebo. A problem analysis report is generated on demand.

PROJECT DESCRIPTION: PAVILION RECYCLING SYSTEM

A small town is interested in developing a system to control a recycling machine for returnable glass, plastic, and metal cans or bottles. The system will physically be located in a centrally located pavilion, near the town hall. The recycling machine can be used by up to three customers at the same time and each customer can return all three types of items. These items come in various types and sizes. The machine must check which type of item was turned in so that it can print a receipt. A receipt, which can be taken to a cashier, will be printed out. The receipt must contain the total value of the items turned in and the value of each item type (glass, plastic, metal).

The machine has to be maintained, so information kept for the maintenance operator must include the total quantity of each item type that has been turned in since the last time the totals were cleared. This information should be able to be printed out. In addition to these totals, the maintenance operator should be able to change the values assigned to individual item types. The machine has numerous mechanical functions which can go awry. The machine has an alarm which indicates that an item is stuck or that the receipt roll is out of paper.

To return items the customer first presses the receipt button to clear all totals. The system then places the items into the correct item type slots. With each item deposited the machine increases the daily totals and the customer totals for that item type. The customer presses the receipt button again to indicate the end of his/her transaction. The action prints the receipt and updates the daily totals.

The operator needs the ability to turn the alarm off, print the daily reports, and clear the report totals. Not only can the value of the items be changed, but because manufacturers regularly change their packaging, the operator must be able to change the allowable sizes for each item type. When items are stuck the customer is prevented from inserting more items but that customers totals are not lost. After the stuck item is cleared from the machine, the customer can continue to insert items which are added to his/her previous totals.

Everything associated with the sorting and validating of the glass, plastic, and metal submitted is done by separate hardware located within the recycling machine. This hardware will determine the type and size of items submitted and provide that information, along with chute number, to the software system.

In addition to the daily report, weekly and monthly reports must be generated. These will be requested by the operator at the end of each day, week, and month, respectively. Each of these reports must be broken down by type of recyclable and chute used, and must also report grand totals.

Software Project Management Plan
Small Project

Week.Class	CI Id	Description
1a		Requirements statement distributed
1b	CI-1	Requirements document: abstract of project and detailed list of requirements
2b	CI-2	Analysis decisions completed, documents delivered: CD, DFD, and data dictionary
3b	CI-3	Design documents delivered: system architecture - structure chart and external descriptions of modules and interfaces
4b	CI-4	Test plan delivered: classes of tests for each requirement
		Presentation of design review Modify design and check coding standards Begin coding system and design of test cases
5b	CI-5	Test cases delivered: specific tests, their input and expected output and their relation to requirements
		Code reviews and unit tests and corrections System testing and corrections to program
6b	CI-6	Documented source code
	CI-7	Executable code
7a	CI-8	Certified Acceptance Test: documentation of test cases and their relation to the test plan and documentation of the consistency of the source code structure with the architectural design; also include package of CI-1 through CI-5
		Presentation of system to customer

NOTE: All presentation/review items are distributed to designated reviewers 24 hours prior to the presentation/review.

TEAM MEETING RECORD

TEAM ID:

DATE: LOCATION: START TIME: END TIME:

PRESENT: ABSENT:

AGENDA

ITEM PERSON RESPONSIBLE

1.

RESOLUTION:

2.

RESOLUTION:

3.

RESOLUTION:

4.

RESOLUTION:

SUMMARY (New issues, difficulties, etc):

TASK LIST

TASK	DUE DATE	PERSON RESPONSIBLE

LAB NUMBER: 002

TOPIC(S) FOR LAB:
Context diagrams
Data flow diagrams
Leveling
Balancing

INSTRUCTIONAL OBJECTIVE(S):

1. Identify external entities, system inputs, and system outputs from a problem specification in the form of a context diagram.
2. Develop the first level of a data flow diagram from a context diagram.
3. Verify the balancing of a data flow diagram with a context diagram.

ASSOCIATED LECTURE NUMBER:
Lecture 004

SET UP, WARM-UP:

During the associated lecture session the students were introduced to the concepts of context diagrams, data flow diagrams, leveling, and balancing. During this class presentation the students, as a group, examined several examples of context diagrams and data flow diagrams as well as developing ones from a problem specification. A problem of a similar level is presented here for the small project teams to attempt, as a group, to develop a context diagram and first-level data flow diagram.

PROCEDURE:

1. HANDOUT - A narrative description of a client request
   The students, separated into their project teams, are handed a brief description of a problem specification and are given 10 minutes to develop a context diagram.

2. The class is reconvened as one group. The instructor draws the circle representing system in the context diagram on the board and solicits inputs, outputs, and external entities from the various teams. This is used to discuss the scope of the problem (e.g., what is included and what is excluded from the system) and to clarify the problem description. The result is a context diagram to be used as input to the next step in this lab.

3. Steps 1 and 2 are repeated to develop a first level data flow diagram.

ASSOCIATED HANDOUTS:
Narrative description of a client request

EXAMPLE OF CLIENT REQUEST - MATCH MAKING SERVICE

I run a match-making service that I want to automate. Here's how it works. I guarantee people who subscribe to my service that I will provide them with the names of three compatible persons of the opposite sex whenever they request with the following two conditions:

(1) they can't request names more than once a month; and
(2) they can't expect all three names provided to be new (since their previous requests) unless they have contacted and talked to those provided on previous lists. (I.e. any name on a previous list will continue to appear on new lists unless they have contacted and talked to that person and reported it back to me).

I build a profile for each subscriber. The profile includes such things as personality traits, interests, intellectual level, education, age, values, etc. The profile is based on information obtained from three sources: a detailed questionnaire completed by the subscriber; a personal interview with the subscriber; and a questionnaire completed by each of three references whose names are provided by the subscriber.

People who want to subscribe to the match-making service are not automatically accepted. A review of applicant is conducted based on the questionnaire, references, and medical history. Based on the review, applicants are notified of their acceptance or rejection.

EXAMPLE OF CLIENT REQUEST - SMALL COLLEGE BOOK ORDERING

The following describes how the bookstore at a small private college manages the ordering of textbooks.

The bookstore maintains an inventory card for each course in the college catalog. Each inventory card contains the title, author, and publisher of the textbook currently used. It also contains the number of the textbooks that are already in stock.

Midway through the spring semester, each academic department provides the bookstore with textbook information for each course they will be offering in the next academic year. The information provided is title, author, and publisher of textbook to be used, and the expected course enrollment.

The bookstore then creates a Books Needed File containing the title, author, publisher, and number needed (expected enrollment minus the number in stock) for each book to be used in the next academic year.

During the last week of the spring semester the bookstore will buy books from students if the Books-Needed-File indicates a need. Of course, each time a used book is purchased, appropriate updates are made in the bookstore's records.

Over the summer the bookstore prepares an Order-List containing the title, author, publisher, and number to be ordered for each book that is still needed. The Order List is then used to create an individual Book Order Form for each publisher. These Book Order Forms are sent to the publishers.

EXAMPLE OF CLIENT REQUEST - STUDENT GOVERNMENT ELECTION

As the advisor to the Student Government Association (SGA), I would like an automated election system developed. The system could then be used for SGA elections, various referenda, and other types of campus elections (for example, balloting for homecoming king and queen) that are conducted by student organizations.

I would like to be able to use the system to create the ballot, to conduct the voting, and to report results. Typical SGA elections consist of several types of votes. First, there are several offices for which all eligible voters can vote for one exactly candidate (e.g., for offices such as President, Vice-President, Secretary, etc). Second, senators are elected to represent a particular school or college and only students majoring in that school or college are eligible to vote. For example, the College of Arts and Sciences may have 7 senate seats and 12 candidates. In that case, students majoring in a department within Arts and Sciences would be able to vote for up 7 candidates from a list of 12. Similarly College of Business majors elect senators to represent business, and so on for each school or college. Incidentally, there are also a number of "at large" senate seats for which all students vote. Third, there may be issues placed on the ballot.

I envision a system in which we would strategically place PC's to serve as "voting booths" at several campus locations and students would use these to cast their votes. The ballot would have previously been created and placed in these machines.

LAB NUMBER: 003

TOPIC(S) FOR LAB:
Requirements list for small project
Context diagrams
Data flow diagrams
Leveling
Balancing

INSTRUCTIONAL OBJECTIVE(S):

1. Clarify requirements for small project.
2. Identify external entities, system inputs, and system outputs from a problem specification for the team's small project in the form of a context diagram.
3. Develop the data flow diagram flows for the team's small project.
4. Verify the balancing of the data flow diagrams and context diagram.

ASSOCIATED LECTURE NUMBER:
Lecture 005

SET UP, WARM-UP:

During an earlier lecture session, the students were introduced to the concepts of context diagrams, data flow diagrams, leveling, and balancing. The students, as a group, considered several examples of context diagrams and data flow diagrams as well as developing ones from a problem specification. The students also worked individually on a problem of a level similar to their first team project. The possible answer was discussed during class. The teams, on their own, begin developing these same diagrams for their small project.

PROCEDURE:

1. The students meet in their individual teams to discuss, with their customer, the abstract and requirements list which they have previously submitted to their customer. This interaction is intended as a preliminary review to make sure that the requirements list is an adequate first draft (i.e. to assure they are on the right track and, if not, to redirect them). If there are multiple small project teams then other faculty can act as customers for the projects.

2. Based on these documents and their discussion with their customer, the students are given the rest of the lab period to begin work on their context diagram and data flow diagrams for the small project. The customer does not meet with the student team during this time but is readily available should questions arise.

ASSOCIATED HANDOUTS:

LAB NUMBER: 004

TOPIC(S) FOR LAB:
Structure charts
Coupling
Cohesion
Fan-in, Fan-out

INSTRUCTIONAL OBJECTIVE(S):

1. Clarify understanding of notation and content of structure charts

ASSOCIATED LECTURE NUMBER:
Lecture 006

SET UP, WARM-UP:

During an earlier lecture session, the students were introduced to the structure charts as a design representation, including the notation used and the information conveyed in structure charts. Also discussed were coupling, cohesion, fan-in, and fan-out as design criteria. As a followup, we're going to examine some structure charts to verify your understanding.

PROCEDURE:

1. Provide a well-designed structure chart and briefly explain the system depicted. Examples are plentiful, including the following:

   a. Mynatt, pp. 161 - Subscription system
   b. Mynatt, pp. 165 - Concordance system
   c. Schach, pp. 295 - Count words in a file
   d. Conger, pp. 299 - Master file update
   e. Kendall, pp. 348-353 - Pay invoice system
   f. Eliason, pp. 466-467 - Enter customer payments

2. Ask a series of specific questions aimed at clarifying the notation and terminology of structure charts. These should cover hierarchical issues, notation for data and control couples, fan-in and fan-out counts, and naming of components.

3. Ask a series of discussion questions to illustrate how one can examine a design through the structure chart. Include discussion of interfaces (coupling and cohesion).

ASSOCIATED HANDOUTS:
Structure chart example

LAB NUMBER: 005

TOPIC(S) FOR LAB:
Preliminary design (structure chart, external module description)

INSTRUCTIONAL OBJECTIVE(S):

1. Begin development of CI-3, a structure chart and external module descriptions, for the small team project.

ASSOCIATED LECTURE NUMBER:
Lecture 007

SET UP, WARM-UP:

Recall that we described requirements analysis and specification as extraction processes and as iterative processes. As you proceed in your projects it will continue to be important to interact with your customer, and refine and change the requirements and associated documentation as changes occur. We have looked over your CD, DD, and DFDs (CI-2) recently submitted and noted problems that need to be addressed before beginning your designs. The requirements have to be accurate before you build your design on them.

Then we want you to begin the designs, as illustrated in the lecture, for your small project. Specifically you are to develop a structure chart and external descriptions of the components.

PROCEDURE:

1. Feedback of the first-draft CD, DD, and DFDs are provided to each team (separately) based on a preliminary review, by the instructor, of the functionality and the notation. This interaction also provides a further opportunity for requirements to be refined and understood. It is made clear that any problems identified are to be addressed and incorporated into all documentation.

2. Teams are given the remainder of the period to begin development of their structure chart and external module descriptions (CI-3). The customer/instructors are available for the remainder of the lab period.

ASSOCIATED HANDOUTS:

LAB NUMBER: 006

TOPIC(S) FOR LAB:

Additional feedback on CI-2
Preparation for small project teams' design review presentations

INSTRUCTIONAL OBJECTIVE(S):

1. Understand suggestions to improve and/or correct teams' first drafts of CD, DD, and DFDs submitted previously.
2. Begin preparing teams for design review presentations.

ASSOCIATED LECTURE NUMBER:

Lectures 007 and 008

SET UP, WARM-UP:

As you further refine the project requirements, interaction with your customer and one another continues to be critical. As changes and refinements occur, it is also critical to modify all CIs to reflect the current system. We have done a more thorough review of your CD, DD, and DFDs and want to discuss our findings so that you can incorporate our suggestions into your model.

In addition, we want to prepare you for the upcoming design review presentations. As a team, you have spent a good deal of time extracting and understanding your customer's needs and attempting to specify them clearly and completely. At this point it is in the best interests of both you and your customer to have that work more formally reviewed for the purpose of improving it and to assure that it is complete and correct. That is the intent of the design review. In preparation, we are also going to review some general procedures for reviews and point out some common pitfalls.

PROCEDURE:

1. Detailed feedback on the first-draft CD, DD, and DFDs are provided to each team (separately). Careful attention has been given to both notation and functionality. This interaction also provides a further opportunity for requirements to be refined and understood. It is made clear that the corrections are to be incorporated into all documentation.

2. The predominant method of design evaluation is the design review. Typically design reviews are conducted at several points in the design process. At each review, the basic questions to keep in mind are:

   (1) Does the design fulfill the requirements?

   (2) Does the design meet established design standards (quality, maintainability, cohesion, coupling, reuse, testability, ...)

To show that the design fulfills the requirements, the reviewer can go through the requirements list, one-by-one, and assure that the design meets the requirement. How effective this is depends on the quality of the requirements list.

3. Discuss the following regarding the formal design review presentations.

   a. The purpose of the review is to improve the product. All participants (developers, customers, and other reviewers) have a common goal: to identify problems that need to be addressed and to come out with clear and complete requirements that, if met, will satisfy the customer. This is a <u>quality assurance activity</u>.

   b. A common tendency of the development team is to <u>defend</u> their work and, in so doing, to <u>resist</u> change. It is understandable why this occurs but clearly it is contrary to the purpose of the review. It is much easier and cost effective to make changes now than later.

   c. Another manifestation of this problem is the "that wasn't in the specs" response. It's unrealistic to expect the specs to be complete. As the maintenance lecture and readings demonstrated, most defects can be traced to requirements problems. Your job is to <u>extract</u> requirements and then improve them if necessary.

   d. Reviews are to identify problems <u>but not to solve them</u>. Resist the urge to come up with solutions (to hack) during the review, or to let others come up with solutions. Note the issue and assure the customer that it will be addressed if possible.

   e. The customer is the person whose needs must be met and the customer is the person who is going to pay you. You can't insult the customer by implying that you understand his/her needs better than he/she does. (Even if you do, and you probably don't.)

   f. Sometimes the best answer is "I don't know" or "we didn't consider that". It's unusual <u>not</u> to have some requirements that were overlooked or misunderstood. Assuring the customer that the issue will be addressed (and subsequently addressing it) is sufficient.

      Bluffing, or conning, to "cover yourself" is dangerous. You are unlikely to fool everyone and you risk damaging your credibility. One of the things happening at these reviews, particularly early ones, is that you are building a rapport with the customer; you are establishing credibility.

   g. This is a <u>team</u> effort. A team working on a project is different from a group where each individual is working on different things. The customer should see a team working on "our project" rather than a group of individuals, each working on "his/her" part. All teams members should be knowledgeable about the entire project, but each will have more detailed knowledge in particular areas. Cooperating in responses, deferring a question to the appropriate person, is appropriate. Demonstrate that you are a team.

f. Plan the presentation ahead of time. Don't wing it. Plan who will do what, and when. Then make a dry run of the entire presentation ahead of time. This is necessary even if each person has his/her part well prepared. To do otherwise is analogous to doing unit (module) testing and not doing integration testing.
   This will eliminate timing problems and smooth transitions in the presentation. A common problem is abrupt transitions.

4. HANDOUT - Preliminary design review form
   Distribute and discuss preliminary design review form. Each student will utilize this form as a reviewer (in reviewing the other team presentations.)

5. HANDOUT - Suggestions for giving and oral presentation
   Distribute and discuss suggestions for giving an oral presentation. These suggestions are for an oral presentation in general but are directly applicable to the team presentations that will occur throughout the course.

6. HANDOUT - Oral presentation evaluation form
   Discuss the evaluation form. This will be used by reviewers to provide feedback on all presentations in the course.

7. Remind students that material to be reviewed must be provided to reviewers in advance. Make arrangements for the materials to be provided to the instructors for duplication and then made available to the reviewers.

<u>ASSOCIATED HANDOUTS:</u>

Preliminary design review sheet.
Suggestions for giving and oral presentation
Oral presentation evaluation form

Preliminary Design Review Form

Project Name	______________________
Reviewer Name	______________________

I. High Level Issues

A. Requirements: any requirements missed, requirements over-worked?

B. Design : suggestions for improvement of architecture or procedures; other strategies

II. Design Deliverable Details

A. Test Plan: items over-tested or under-tested, suggested tests

B. DFD: good use of notation, clear model, suggested improvements

C. Comments on other deliverables

Suggestions For Giving An Oral Presentation

1. In preparation, concentrate on "why" rather than "how" questions. Why am I giving this talk? Why is this audience interested in this topic? Ask yourself:

   A. What is my purpose? (For example, am I reporting on a topic, am I arguing a case, am I trying to change their opinion, etc.?)
      (1) Why am I talking to this audience about this topic?
      (2) What do I want them to know, think, do, or feel as a result of my talk? Do I want to change their opinion on the subject?
      (3) What must I do in my presentation to achieve this?

   B. Who will be listening to me? (I.e. who is the audience?)
      (1) How much do they know about my topic? (Don't always assume that they're already familiar with the topic.)
      (2) What is their attitude about my topic?

2. Prepare and then <u>rehearse</u> the presentation. Don't "wing it". Lack of preparation is usually obvious to the audience. Get a friend to watch you rehearse the presentation or rehearse in front of a mirror.

3. Check out the room ahead of time. Make sure you know how to operate any equipment you'll be using and make sure it's working properly.

4. You'll usually have a specific amount of time allotted for the presentation. Use it wisely. Organization and format are critical, especially when the allotted time is short. Be sure, through rehearsing, that your presentation fits the time allotted.

5. Consider this outline for your presentation.

   A. INTRODUCTION - Give the title of the presentation, your name, and the names of anyone else involved. (For example if you are presenting the work of a team, identify the team members. If you are presenting a review of an article, identify the author and source of the article.)

   B. OVERVIEW
      (1) Purpose and scope - Don't assume the audience is already familiar with the topic. Give a brief description (i.e. an abstract). Jumping immediately into details will quickly lose the audience. This is the time to give your audience a reason to listen to you.
      (2) Outline of presentation.

C. BODY OF PRESENTATION - Have a specific but limited number of points to cover; don't try to do too much. Make a smooth, logical transition between points as well as between the introduction and overview, the overview and the body, and the body and the conclusion.

D. CONCLUSION - Summarize key points, findings, or recommendations.

E. QUESTIONS - There may be a planned question and answer period to follow the presentation. If not, allow a few minutes for questions. Don't be intimidated or defensive; usually the questioner is genuinely interested and may even be helping emphasize or clarify a point. Answer as best you can and don't be afraid to say "I don't know".

6. Use the speaking medium to its best advantage. Remember you are giving a talk, not a written report. Use the strengths of oral speech. Give the big picture; explain rationales; motivate the audience. Oral communication is a much more natural, personal, human activity than is written communication. <u>Talk to</u> and <u>look at</u> your audience.

7. Avoid technical jargon unless you're sure it is familiar to the audience. Use simple straightforward sentences. Explain clearly the real meaning of any statistics, numbers, charts, or graphs that you use.

8. Include your own opinions, observations, or perceptions. Personalize the topic to your own (and/or the audience's) common experiences.

9. If appropriate use visual aids to enhance the presentation but remember that they are <u>aids</u> to your talk and should not simply display the same words you're speaking. They can be overhead transparencies, chalk/chalkboard, handouts, slides, computer demonstration, or combinations of these. Visual aids can support, enhance, clarify subject matter and to focus attention on major points.

   They must be visible to the audience; they're not effective if your audience can't see them. This is a common mistake in using a computer demonstration in which the screen can be seen by only part of the audience or when the print on overhead transparencies is too small or too light. Make sure your visual aids are legible and large enough to be seen.

   Make the message of each visual aid clear. Beware of including too much. Keep them simple.

   Don't read the visual aids; use them to focus attention on key points. One of the quickest ways to lose your audience is to read to them instead of speaking to them.

10. Most presenters are nervous but it doesn't have to hurt the effectiveness of the presentation. The audience will tolerate nervousness and, in fact, will tend to "pull for you". They won't tolerate your disinterest or lack of preparation. Give the audience the sense that you've got something to say; that you want to be there. Some suggestions to help deal with your nervousness:

    A. Prepare.

    B. Maintain eye contact. Look at and speak to the audience.

    C. Move around as you speak; use some expression. Come out from behind the podium. Don't lean on the podium or sit on the desk. Show that you're interested.

    D. Avoid nervous or annoying mannerisms and expressions. Rehearsing the presentation will help reveal these.

    E. If you use written notes then put them on index cards. They're less obvious and avoid the effect of "nervous hands" shaking your notes and distracting the audience.

11. Dress appropriately. If you're not sure what is appropriate then find out ahead of time. It's possible to be over-dressed and it's possible to be under-dressed.

ORAL PRESENTATION EVALUATION FORM

PRESENTER ______________ **EVALUATOR** ______________

Use the scale below to rate the presentation on Organization, Delivery, Content, and Overall Effectiveness. The following qualities should be present for an ABOVE AVG to EXC rating.

ORGANIZATION
* Obviously well prepared, organized
* Intro includes (1) name of presenter & others involved (project team, committee, advisor, etc); (2) purpose; (3) brief overview
* Body of talk covers main ideas; smooth transitions between points
* Concludes with definite ending; summarizes main points
* Uses time well, stays within allotted time frame (excluding questions)

DELIVERY
* Obvious knowledge/understanding of subject
* Talks (not reads) to audience; looks at audience
* Poised; able to control nervousness; no distracting mannerisms; good posture; natural movement; appropriately dressed
* Talks clearly, easily understood, good grammar & pronunciation

CONTENT
* All requirements specified in particular assignment met
* **Essence** of talk clearly conveyed to, understood by audience
* Supporting materials and visual aids, if used, enhanced presentation; were easy to read/understand by all; were effective
* Questions handled well

Mark an **X** to indicate your rating. The scale ranges from unsatisfactory (UNS) on the left to excellent (EXC) on the right.

ORGANIZATION RATING

```
        BELOW          ABOVE
UNS      AVG     AVG    AVG     EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

DELIVERY RATING

```
        BELOW          ABOVE
UNS      AVG     AVG    AVG     EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

CONTENT RATING

```
        BELOW          ABOVE
UNS      AVG     AVG    AVG     EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

OVERALL RATING

```
        BELOW          ABOVE
UNS      AVG     AVG    AVG     EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

Please use other side for additional comments.

LAB NUMBER: 007

TOPIC(S) FOR LAB:
Test plans

INSTRUCTIONAL OBJECTIVE(S):

1. Develop CI-4 (classes of tests) for small team project.

ASSOCIATED LECTURE NUMBER:
Lecture 009

SET UP, WARM-UP:

In the testing lecture we distributed and discussed a preliminary test plan for the KoFF Video Rental System. A key feature was the traceability between tests and requirements. We want you to develop a similar test plan for your small projects; in particular, the equivalent to section 4.0 (Figure 3.5-1) of the Koff preliminary test plan. The test plan you develop will form the basis for acceptance testing of the system.

PROCEDURE:

1. Refer back to the Koff Preliminary Test Plan. Specifically review:
   a. Section 3.1 (Requirements Traceability) and Figure 3.2-1 (Test/Requirement Traceability Matrix);
   b. Section 4.0, Figure 3.5-1 (Tests to be performed).

   In this example, first discuss the development of categories of tests. Then discuss the development of items to be tested within each category, followed by the ordering of the tests and prerequisites for each test.

2. Teams are given the remainder of the period to meet and begin development of a test plan (CI-4) for the small project. Their customer and the instructors are available for the remainder of the lab period.

ASSOCIATED HANDOUTS:
Preliminary test plan for KoFF system (distributed in associated lecture)

LAB NUMBER: 008

TOPIC(S) FOR LAB:
Design review team presentations for small projects

INSTRUCTIONAL OBJECTIVE(S):

1. Present requirements, design, and test plans for review.

ASSOCIATED LECTURE NUMBER:
Lecture 010

SET UP, WARM-UP:

Earlier we discussed the purpose and general procedures of formal reviews. Today we are all going to participate, each in multiple roles, in preliminary design reviews for your small projects. Each of you will participate as a member of a team whose work is being reviewed and as a reviewer for the other teams. Remember that the purpose of the review is to improve the quality of the software system under development.

PROCEDURE:

1. a. Remind teams that during their review they should note any issues which arise that require attention. Each item on this "issues list" must be addressed and appropriate modifications made where needed. The issues list thus serves as action item checklist for the team as they addresses the issues.

   b. The instructor should maintain his/her own issues list as a means of establishing a follow-up procedure to assure that the items are addressed.

2. Determine the order of the team presentations and begin the reviews.

   Instructors should maintain their role as customer as much as possible, reverting to role of instructor only when necessary for such things as maintaining the schedule, reminding participants of the purpose and/or ground rules, and maintaining order. Critiques should be saved until the next lecture or lab.

ASSOCIATED HANDOUTS:
Material to be reviewed has been provided to reviewers in advance.

LAB NUMBER: 009

TOPIC(S) FOR LAB:
Feedback on design review presentations.

INSTRUCTIONAL OBJECTIVE(S):
1. Provide timely feedback on recent design review presentations.
2. Provide timely feedback on all configuration items for small projects submitted to this point.

ASSOCIATED LECTURE NUMBER:
Lecture 011

SET UP, WARM-UP:

You recently experienced your first design review presentation with your customer and other reviewers. We want to provide you with our reactions to the presentations.

PROCEDURE:

1. With all teams present, provide general information pertaining to customer reviews in general. Defer specific comments on their presentations for individual meetings with each team. It is helpful to review the guidelines presented in Lab 006 and relate them to specifically to their reviews.

2. Meet separately with each team to provide specific feedback on the items reviewed (requirements list, CD, DD, DFDs, structure chart and interface descriptions, and test plan). Specifically ask about their "issues list" which should have been compiled during the review. (Use your own issues list as a check.) Ask how each item was addressed and the disposition of each.

   This meeting is critical to establish baseline requirements and design. Teams are about to begin implementation and agreement must be reached on what is to be implemented; in a sense the requirements and design are being frozen (after the necessary changes based on the review are made). An early deadline needs to be placed on teams for making the necessary revisions and baselining the configuration items.

ASSOCIATED HANDOUTS:

LAB NUMBER: 010

TOPIC(S) FOR LAB:

Feedback on CI-5, test plans and test cases.
Small project team preparation for team acceptance test presentations.

INSTRUCTIONAL OBJECTIVE(S):

1. Provide timely feedback on CI-5.
2. Prepare teams for acceptance review presentations of small projects.

ASSOCIATED LECTURE NUMBER:

Lecture 012

SET UP, WARM-UP:

Soon your team will be meeting with your customer to conduct acceptance testing. Today we want to review your test plans and test cases in preparation for an acceptance test with the customer.

PROCEDURE:

1. Distribute copies of each team's CI-5 to the class. Discuss each of these. Pay particular attention to whether all requirements are completely tested.

2. HANDOUT - Acceptance test review
   Distribute and discuss acceptance review checksheets.

   Discuss how their test review presentations will differ from a normal acceptance test. In-class acceptance testing cannot meet all of these conditions, but certainly a simulation can be done which at least meets conditions 2 and 4. Documentation can be presented so that the maintainability (condition 3) is clear.

   The product demonstrated should be as close to the product to be delivered as possible. Discuss the potential problems caused by embedding test scaffolding or a test harness into the software being demonstrated. This should be avoided wherever possible, and where necessary the scaffolding should be removed when the system is delivered to the customer. Point out that this could mean that the system on which acceptance testing is conducted is a different system that the one delivered.

3. Stress that these tests will form the basis for acceptance testing.

ASSOCIATED HANDOUTS:

Acceptance test review

Acceptance Test Review

The Acceptance test is normally the final test before customer acceptance of a completed and proven product. There are several conditions that characterize such a test.

1. It is generally done with customer supplied data and under supervision of the customer organization.

2. The customer is concerned with the ease of use of the system and the ease of training other users.

3. The customer wants a system which is easy to maintain. Sometimes final approval requires the approval of the maintenance organization.

4. The customer's primary question is if the system meets the specified requirements. This is most easily shown by publicly executing the scenarios of the test plan.

5. The customer is interested in more than just the software. They are interested in all of the deliverables including tested user manuals and training manuals, maintenance and update information.

6. Programs that work on one machine may not work on another. It is important to try to test in an environment similar to the one in which the system will be installed?

In-class acceptance testing cannot meet all of these conditions, but certainly a simulation can be done which at least meets conditions 2 and 4. Documentation can be presented so that the maintainability (condition 3) is clear. The product demonstrated should be as close to the product to be delivered as possible. Do not embed test scaffolding or a test harness into the software being demonstrated.

projects is that the instructor has other faculty or administrators available as information resources.

c. Management of Teams.

i. General Guidelines

There are several guidelines to the development and management of team projects.

1. **Carefully select the project.** Selecting a project that is too simple allows the students to succeed in spite of sloppy development (i.e., hacking) and does not give them an appreciation for the complexity of software development. Selecting a project that is too complex presents the students with an unmanageable task. Student satisfaction with the learning experience is greater when they deliver a functioning system.

2 **Carefully separate the roles of instructor and customer.** Regardless of whether the customer is real, is simulated by someone other than the instructor, or is role-played by the instructor, the roles should remain separate. The customer has domain expertise and understands his/her own needs but does not necessarily have computer science expertise. The instructor has computer science expertise but does not necessarily have domain knowledge or understand the customer's needs. During interactions, students should always be aware of who is responding - the customer or the instructor. When role-playing, a simple yet effective way to make this visibly apparent is for the instructor to have an instructor hat and a customer hat (baseball caps work particularly well) and to always remember to wear the appropriate hat. A less visually apparent method is to simply preface statements appropriately (e.g., "as the customer, ...").

3. **Provide appropriate access to the instructor and to the customer and facilitate interaction between students and customer.** The level of accessibility will be different for the instructor and the customer. Access to the instructor is typically open. Students also need extensive access to the customer, particularly during requirements elicitation, but that access should be more limited. Meetings with the customer need to be scheduled, as they would be in the real world.

4. **Use team projects rather than individual projects.** Team projects require communication between team members as well as with the instructor and customer and better reflect the real world. The number of projects to be managed simultaneously by the instructor is reduced. Teams of four to six students are appropriate.

5. **Solve the problem before assigning it.** Prior to assigning the project, the instructor should work out a solution, at least through design. This will assure that the instructor is familiar with the problems that the students will face during development.

6. **Provide appropriate steering to student teams.** While the students should define the requirements and develop a solution on their own, appropriate steering away from known pitfalls or overly complex solutions will increase the chances for success. Steering students away from premature concentration on implementation details is often necessary.

7. **Require deliverables other than the source code.** Appropriate deliverables such as requirements documents, design documents, and test plans and results should be required of the students. Otherwise, students can resort to old habits of concentrating strictly on implementation.

ii. Peer and Project Evaluation

Managing the Small Project Teams

In the small project a single team of students works on all phases of the software development. For this type of team structure, a peer evaluation only has to address intra-team issues. Because of the relatively short duration of this project, we only use a peer evaluation at the end of the project. Thus, in this case peer evaluations do not help to identify team issues such as some students failing to carry their share of the load. These are, however, identified by team meeting reports and other interactions with students and student teams during the project. In the peer evaluation we have found it useful to ask each student about their own as well as every other team member's contribution. Students are less prone to exaggerate their own performance and accomplishments if they are aware that other students are also describing their work. This question also helps, when students understate their own contribution to the project. In general we have found the question about what sequence they would re-hire their own teammates to be a useful question doing most stages of development. A sample peer evaluation follows.

SMALL PROJECT PEER EVALUATION

GAZEBO TEAM: PEER/SELF EVALUATION **NAME:** ________________

—

Your responses are confidential and will be seen only by the instructors. Be completely honest. Use back for additional comments.

1. Evaluate the performance of each team member, including yourself with respect to each of the following questions by indicating **SA** (strongly agree), **A** (agree), **D** (disagree), or **SD** (strongly disagree).

	STUDENT					
	1	2	3	4	5	6
He/she took a fair share of the responsibility and work.						
He/she took a leadership role.						
He/she kept aware of the project's problems and progress.						
He/she is knowledgeable of the tools and techniques used.						
He/she attended meetings and cooperated with rest of team.						
He/she gave an honest effort and completed tasks on time.						
I would choose to work with him/her on another project.						

2. Complete Columns A and B for each team member, including yourself.

COLUMN A: Enter +, =, or - as follows.

+ means this person made a significant contribution to the team and should be given a bonus; their individual project grade should be higher than the team grade.

= means this person did their share; their individual project grade should be equal to the team grade.

- means this person's performance was less than adequate and his/her individual project grade should be lower than the team grade.

COLUMN B: Describe his/her major contributions.

TEAM MEMBER	(A) + = -	(B) MAJOR CONTRIBUTION(S)
Student Name1		
Student Name2		
Student Name3		
Student Name4		
Student Name5		
Student Name6		

3. For each item below, rate your team's performance and deliverables produced. UNS represents unsatisfactory and EXC represents excellent.

(a) Interaction with user in understanding/defining requirements

```
        BELOW               ABOVE
UNS      AVG       AVG       AVG       EXC
|-------|-------|------|------|------|------|------|------|
```

(b) Configuration Item 1 - narrative description (abstract) of project and requirements list.

```
        BELOW               ABOVE
UNS      AVG       AVG       AVG       EXC
|-------|-------|------|------|------|------|------|------|
```

(c) Configuration Item 2 - analysis documents: context diagram, leveled data flow diagrams, data dictionary.

```
        BELOW               ABOVE
UNS      AVG       AVG       AVG       EXC
|-------|-------|------|------|------|------|------|------|
```

(d) Configuration Item 3 - design documents: system architecture (structure chart and external description of modules and interfaces).

```
        BELOW               ABOVE
UNS      AVG       AVG       AVG       EXC
|-------|-------|------|------|------|------|------|------|
```

(e) Configuration Items 4 and 5 - test plan (classes of tests for each requirement); test scenarios (specific tests, input, expected output, etc.)

```
        BELOW               ABOVE
UNS      AVG       AVG       AVG       EXC
|-------|-------|------|------|------|------|------|------|
```

(f) Configuration Items 6 and 7 - documented source code; executable.

```
        BELOW               ABOVE
UNS      AVG       AVG       AVG       EXC
|-------|-------|------|------|------|------|------|------|
```

(g) Configuration Item 8 - documentation of testing; acceptance testing plans, documentation, etc.

```
          BELOW              ABOVE
UNS        AVG       AVG      AVG       EXC
|-------|-------|------|------|------|------|------|------|
```

(h) Configuration Item 7 - Acceptance test review

```
          BELOW              ABOVE
UNS        AVG       AVG      AVG       EXC
|-------|-------|------|------|------|------|------|------|
```

(i) Overall, rate the your team's performance for the entire project?

```
          BELOW              ABOVE
UNS        AVG       AVG      AVG       EXC
|-------|-------|------|------|------|------|------|------|
```

(j) Overall, the tools team and the materials they have produced have been

```
NO            LITTLE                                        MUCH
HELP            HELP             OK           HELP          HELP
|---------|---------|---------|---------|---------|---------|---------|---------|
```

4. If you had to do this project again and were in charge of hiring personnel, in what order would you rehire the team? In other words, who would be the person on your team you would rehire first, second, third, etc.? (Be sure to include yourself).

   1.

   2.

   3.

   4.

   5.

   6.

SMALL PROJECT EVALUATION

In addition to a peer evaluation, we use a project evaluation form. We do not comment on a particular students contribution on this form. We comment on the quality of the delivered product giving a clear grade for the quality of the product. We also include an individual student grade for their contribution to the project. This grade may differ significantly from the product grade. A sample of a completed small project evaluation follows.

PROJECT 1 EVALUATION: FIRE AND SECURITY ALARM SYSTEM

TEAM MEMBER: ____________________

TEAM PRODUCT GRADE: ___ **TEAM MEMBER'S GRADE:** ___

COMMENTS ON DELIVERED PRODUCT

These comments pertain to the delivered software product and are not necessarily reflective of the time and/or effort expended.

OVERALL PACKAGING

Product gives appearance of having been thrown together up to the last minute, including some items being crossed out and others being penciled in.

NARRATIVE DESCRIPTION

In paragraph 7, "same span of time .... " is still too vague to be tested.

REQUIREMENTS LIST

#9 and Footnote are inconsistent with one another.

#13, #15: indentation erratic.

#20: incident reports and frequency reports never fully defined.

CONTEXT DIAGRAM, DFDs, DATA DICTIONARY

Many items missing from data dictionary, including:

- Notify
- Incident report
- Frequency report
- Incidents
- Signal to Fire Equipment
- Signal to Warning Device
- Signal to Lock/Unlock
- Room Function

No data stores defined in data dictionary.

Use of "Flag" in data dictionary is still awkward. It would be much more meaningful to do something like the following:

Hazard level = High | Normal
Type = Fire | Security
Alarm condition = In-service | Out-of-Service
etc.

In some cases, more meaningful names could easily be used; for example Type is still too vague.

Context diagram and DFD are not balanced.

DFD is rough; far from form expected in finished product.

No leveling of DFD.

Transform 1 has no output.

Transform 2 has no input.

As shown in model, SetUpFile should be an external entity.

DESIGN DOCUMENTS

No external description of modules and interfaces submitted.

Various inconsistencies or omissions in structure chart; for example Get Info (page 2) doesn't return any information.

Module (and procedure) names should always be as descriptive as possible and consist of Verb and Object.

CODE

Design and code are inconsistent.

In places it is tough to distinguish between test modules and product modules; for example the OurTime procedure.

Programming standards were not followed in several aspects including identifier dictionaries, input and output descriptions, use of meaningful identifier names (for example, look at TimeCompare and TimeSubtract).

Inconsistent documentation blocks (for example, none on Init and CallProcedures).

Comments - look at II.7 of programming standards.

What is purpose of procedure CallProcedures?

TEST PLAN, TEST RESULTS

Test categories too broad; for exam[illegible] consider Section C (Alarm Responses) - these need to be broken dow[illegible]rther to adequately test system.

Test procedure form: (a) all look like low-level unit tests; (b) all end with the generic statement "Verify test data is output to test result file." Should also worry about correctness of output.

Test results somewhat confusing and appear inadequate; not mappable to test procedures.

Managing the Extended Project Teams

The extended project introduces some new complications in managing projects. First, since it is a multi-semester project there may be new students joining the project as well as students leaving the project at its mid-point. The problems of training and familiarizing the new students with the project are reduced here by having all milestone information ready for the second semester of the project. There is a virtue to the second semester. It provides the opportunity to use the information you learned about the student team's dynamics in the first semester. Using this information, teams can be restructured to reduce personal conflicts in the second semester, and to maximize the use of individual student skills..

The extended project introduces a new management problem, intra-team dependency. There are several techniques to facilitate communications between teams. One tool we use is to appoint a team liaison to other teams. It is critical to project success that the teams work together toward a common goal. Sometimes when problems arise there is a tendency for the teams to compete against each other or blame each other for problems. In managing an extended project, one must examine the status of the project at several points to help keep it on track. There should be at least two peer evaluations; one at the semester break and one at the end of the project. There are numerous opportunities for product evaluation, e.g., reviews and inspections, comments on milestone documents, etc. These reviews are directed at individual teams. Sometimes we have used a composite evaluation document, so that each team was made aware of the strengths and weaknesses of their own work as well as others team's work. Because student's function in a variety of roles on a variety of teams, it helps avoid confusion to place the students names on the form for each team.

The extended project peer evaluation is more complicated because of the intra-team communications. A mid-project and end of project peer evaluation form for a student project called "Third Eye" follow. This was an automated plagiarism detection program.

THIRD EYE PROJECT MID-POINT EVALUATION

Responses are confidential and will be seen only by the instructors. Be completely honest in rating the following from your perspective. In the scale used, UNS represents unsatisfactory and EXC represents excellent. Use back for additional comments.

Configuration management plan

```
      BELOW            ABOVE
UNS    AVG      AVG     AVG      EXC
|------|------|------|------|------|------|------|------|
```

COMMENTS:

Configuration manager

```
      BELOW            ABOVE
UNS    AVG      AVG     AVG      EXC
|------|------|------|------|------|------|------|------|
```

COMMENTS:

Requirements

```
      BELOW            ABOVE
UNS    AVG      AVG     AVG      EXC
|------|------|------|------|------|------|------|------|
```

COMMENTS:

Users manual

```
      BELOW            ABOVE
UNS    AVG      AVG     AVG      EXC
|------|------|------|------|------|------|------|------|
```

COMMENTS:

Test Plan

```
      BELOW            ABOVE
UNS    AVG      AVG     AVG      EXC
|------|------|------|------|------|------|------|------|
```

COMMENTS:

Preliminary design

```
      BELOW            ABOVE
UNS    AVG      AVG     AVG      EXC
|------|------|------|------|------|------|------|------|
```

COMMENTS:

Overall, the Third Eye team

```
      BELOW            ABOVE
UNS    AVG      AVG     AVG      EXC
|------|------|------|------|------|------|------|------|
```

COMMENTS:

THIRD EYE PROJECT MID-POINT EVALUATION

TEAM: REQUIREMENTS * MEMBER: ____________________

Complete the following for each member of the team, including yourself.

In COLUMN A, describe his/her contributions to the project.

In COLUMN B, select **VS** (very satisfied), **S** (satisfied), **D** (dissatisfied), or **VD** (very dissatisfied) to fill in the blank in the statement below.

"I am ________________ with his/her work on the team."

TEAM MEMBER	(A) CONTRIBUTION(S)	(B) SATISFACTION

* A similar form is completed by each member of the other project teams (user interface, test plan, preliminary design, etc.)

THIRD EYE - END OF PROJECT PEER EVALUATION

Responses are confidential and will be seen only by the instructors.

1. Rate the following. UNS represents unsatisfactory and EXC represents excellent.

Configuration management plan

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|----|----|----|----|----|----|----|----|
```

COMMENTS:

Requirements

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|----|----|----|----|----|----|----|----|
```

COMMENTS:

Test Plan

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|----|----|----|----|----|----|----|----|
```

COMMENTS:

Users manual

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|----|----|----|----|----|----|----|----|
```

COMMENTS:

Preliminary Design

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|----|----|----|----|----|----|----|----|
```

COMMENTS:

Detailed design

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|----|----|----|----|----|----|----|----|
```

COMMENTS:

Code and unit test

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|----|----|----|----|----|----|----|----|
```

COMMENTS:

Testing

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|----|----|----|----|----|----|----|----|
```

COMMENTS:

Configuration Management

```
        BELOW               ABOVE
UNS      AVG       AVG       AVG       EXC
|-------|-------|-------|-------|-------|-------|-------|-------|
```

COMMENTS:

Overall, the Third Eye System

```
        BELOW               ABOVE
UNS      AVG       AVG       AVG       EXC
|-------|-------|-------|-------|-------|-------|-------|-------|
```

COMMENTS:

2. Characterize the interactions <u>between</u> the indicated teams using the following scale by circling the most the most appropriate descriptor.

VN = <u>V</u>ery <u>N</u>on-productive **A** = <u>Adequate</u> **VP** = <u>V</u>ery <u>P</u>roductive
N = <u>N</u>on-productive **P** = <u>P</u>roductive

a) Preliminary design team & Detailed design team ----- **VN N A P VP**

b) Detailed design team & Code and unit test team ----- **VN N A P VP**

c) Detailed design team & Testing team -------------------- **VN N A P VP**

d) Code & unit test team & Testing team ------------------ **VN N A P VP**

e) Detailed design team & Configuration manager ------- **VN N A P VP**

f) Code and unit test team & Configuration manager ---- **VN N A P VP**

g) Testing team & Configuration manager ----------------- **VN N A P VP**

THIRD EYE - END OF PROJECT PEER EVALUATION

TEAM: DETAILED DESIGN * **MEMBER:** ________________

Complete the following for each member of the team, including yourself.

In COLUMN A, describe his/her contributions to the project.

In COLUMN B, select **VS** (very satisfied), **S** (satisfied), **D** (dissatisfied), or **VD** (very dissatisfied) to fill in the blank in the statement below.

"I am ______________ with his/her work on the team."

TEAM MEMBER	(A) CONTRIBUTION(S)	(B) SATISFACTION

* A similar form is completed by each member of the other project teams (code & unit test, testing, etc.)

4. Imagine that $2000 in bonuses is to be distributed among the THIRD EYE Project team members. Half of it ($1000) is to be distributed based on the intellectual contribution to the project, i.e., significant ideas and solutions contributed. The other half ($1000) is to be distributed based on amount of individual effort contributed to the project.

   Distribute the bonuses. If you wish, justify each of the assignments. Be very specific; list some especially significant contributions for which the team member should be proud or where the project was made more or less difficult because of it.

Project Member Name	**$1000 concepts**	**$1000 effort**	**JUSTIFICATION**

iii **Regular Team Meeting**

Instead of waiting for students to submit project deliverables to track a project, we require team meeting reports from each team for every team meeting. This allows us to keep track of their work without being intrusive at team meetings. The team meeting report template included on the following page is provided to each team. The team meeting report form is intended to develop a task oriented structure for the meetings. After the first meeting, the tasks assigned at the previous meeting constitute the primary agenda for the next meeting. The status of the work on every task is reported and discussed at the meeting. The problem is either reported as resolved or a new approach is decided upon and assigned. This method of documenting meetings helps to give them a structure and a goal. It also clearly documents individual responsibility for project tasks. There is also a place to report general problems. This is especially useful information to the instructor.

TEAM MEETING RECORD

TEAM ID:

DATE: **LOCATION:**

START TIME:

END TIME:

MEMBERS PRESENT: **ABSENT:**

AGENDA

ITEM **PERSON RESPONSIBLE**

1.

RESOLUTION:

2.

RESOLUTION:

3.

RESOLUTION:

SUMMARY (New issues, difficulties, etc):

TASK LIST

TASK	DUE DATE	PERSON RESPONSIBLE

d. Management of Extended Project

i. Scheduling

It is essential to carefully control both inter-team and intra-team activity if the inverted functional matrix organization is to be effective. Objectives in team management include disciplined system development and overcoming student tendencies to procrastinate and become easily distracted from a task. Practical management techniques for accomplishing these objectives include an effective project start-up, a software project management plan, preliminary and final in-class reviews of team deliverables, regular team status reports, and inch pebbles plans when appropriate.

Assigning all students to teams that begin work at once on the project is important because it sets the tone for the whole project. It is equally important that these teams be coordinated so that each has specific tasks to be undertaken immediately. Thus, the configuration manager, the requirements team, the user interface team, the test plan team, and the tools team are given specific assignments.

The configuration manager is charged immediately with developing a first draft configuration management plan. The configuration management plan, developed by the configuration manager and presented to the class for review, must be in place prior to the development or submission of any other configuration items.

All teams are informed of their specific deliverables. The requirements team immediately begins the process of eliciting requirements from the customer. Similarly, the user interface team interacts with the user and the requirements team to begin establishment of user interface requirements and a format for the preliminary user manual. The test plan team begins work on a test plan shortly after requirements analysis is underway. The tools team develops training materials and documentation for project support. With this start-up strategy, all students are immediately involved in the project.

All projects were carefully scheduled. We have found that unscheduled software is vaporware. Each project has a deliverable schedule which drives the project. These are modified software project management plans.

A software project management plan (SPMP) describes the sequence of activities needed to successfully develop a particular software product. It minimally includes all major milestones and their due dates, the dates when teams are scheduled to begin work, and the list of configuration items (i.e., items to be placed under configuration control). If more detail is desired, the SPMP can also include intermediate targets such as walkthroughs and inspections. Below is a sample small project software project management plan and an extended project SPMP used through preliminary design. To be effective, the plan should be handed out when the project is introduced.

Between deliverables, the plan should be referenced during class and students asked how they are progressing and if they need any help meeting their due dates. Just as the SPMP guides the students through the project, it also helps the instructor in managing the project.

PROJECT 1: SOFTWARE PROJECT MANAGEMENT PLAN

Date	Due	Description
Th 9/2		Customer requirements statement distributed; begin requirements analysis and specification
Th 9/9	Draft CI-1	Narrative description (abstract) and requirements list
Tu 9/14	Draft CI-2	CD, 1st level DFD, and DD
Th 9/16		Begin design; begin test plan
Tu 9/21	CI-1, CI-2	**Requirements review presentation**
Th 9/23	Draft CI-3	Design documents: system architecture - structure chart, external descriptions of modules & interfaces
	Draft CI-4	Test plan: classes of tests for each requirement
Tu 9/28	CI-3, CI-4	Begin design of specific test cases
Th 9/30		Continue modifications based on internal reviews and customer feedback; continue design of test cases
Tu 10/7	CI-5	Test cases: specific tests, their input and expected output and their relation to requirements
Tu 10/12		**Design review presentation** Modify analysis, design, test plan documents based on review; begin coding
: : :		{Internal code reviews, unit testing, system testing}
Tu 10/26	CI-6, CI-7	Documented source code, Executable code
Th 10/28	CI-8	Acceptance Test: documentation of test cases and their relation to the test plan and documentation of the consistency of the source code structure with the architectural design;
		Presentation of system to customer
	CI-9	Deliver "package" of all CI's

——— See notes on attached page ———

NOTES: All Configuration items (CI's) are due at the BEGINNING of class on the specified dates.

All presentation/review items are distributed to designated reviewers 24 hours prior to the presentation/review.

All team members are expected to participate in a meaningful role in at least one of their team's review presentations.

Configuration items:

CI-1	Narrative description (abstract) and requirements list
CI-1	Context diagram, leveled data flow diagrams, data dictionary
CI-3	Design documents: software system architecture - structure chart, external descriptions of modules and their interfaces
CI-4	Test plan: classes of tests for each requirement
CI-5	Test cases: specific tests, their input and expected output and their relation to requirements
CI-6	Documented source code
CI-7	Executable code
CI-8	Documentation of test cases and their relation to the test plan and documentation of the consistency of the source code structure with the architectural design
CI-9	Complete package

Additional details regarding format of CI's will be provided in class.

PROJECT 2: PROJECT SCHEDULE

DATE	CI Id	Description
10/28		Customer request presented.
11/02		Team assignments announced and roles defined.
		Start development of configuration management plan (CMP), preliminary requirements (P_REQ), preliminary test plan (P_TP), and preliminary user's manual (P_UM).
11/09	CI-1	CMP delivered and presentation to teams.
11/11	CI-2	P_REQ delivered and presentation to teams and customer.
11/16	CI-3	P_TP delivered and presentation to teams.
	CI-4	P_UM delivered and presentation to teams.
11/18		Requirements review. Preliminary design begins.
11/23	CI-5	Requirements revised based on review, delivered, baselined.
12/02		Preliminary design review.
12/07	CI-6	Preliminary design delivered and baselined.
12/09	CI-7	Final test plan delivered and baselined.
	CI-8	Final user manual delivered and baselined.
		Entire project package submitted.

NOTES: All Configuration items (CI's) are due at the BEGINNING of class on the specified dates.

All presentation/review items are distributed to designated reviewers 24 hours prior to the presentation/review.

All team members are expected to participate in a meaningful role in at least one of their team's review presentations.

ii. Configuration Management

The SPMP underscores the role and visibility of configuration management. The configuration management plan establishes the place of the configuration manager in the development process. Documents submitted for final review, as specified on the SPMP, are immediately placed under CM and, therefore, configuration control. The first document placed under configuration control is the configuration management plan. A minimum form of configuration control can be established by setting up a directory which holds items placed under configuration control and to which only the configuration manager has write access and other class members have read access.

A good configuration manager makes the whole development process easier. Both the instructor and the configuration manager must promote the configuration management plan. Active and visible support for the configuration manager from the instructor is also necessary. The instructor acts as the configuration control board in handling change requests. The turn-around time on change requests must be minimized to avoid the perception that CM impedes rather than expedites development. Samples of CM plans and change requests are contained in the case study in section V.

e. PROJECT IDEAS

There are sample projects scattered throughout this document. For example, some project examples are contained in Lab001. Additional project ideas are listed below. Many of these were drawn from other software engineering texts.

In selecting projects for a course there are a number of decisions you can make which will help limit the project selection. Some of the generic questions that can be asked are: Is the system one that is intended to be used by someone?, Is it an complete application or is it a piece of a larger one?, What special interfaces are available, and what type of application is it?

Databases are easily understandable types of projects. These systems are specialized data storage and retrieval systems. It is interesting when the algorithms involve concurrency management, retrieval, etc. These projects involve a client server model where the server manages access to some database. The second group of items require very specific customer interfaces.

1. lottery manager
   student Government Association Voting system
   scientific reference library
      personal data management systems
      e.g., wine cellar, record albums, books, videotapes, etc.

polling/voting system
library management system (books & patrons)
store sales activity
genealogical database
software components catalog
photograph library sales system
student record system
student laboratory management system
course scheduling and management system
university department information management
draw diagrams
generate relational database scheme

2. slot machine
vending machine - coin slot & dollar bill slot
electronic banking
electronic mail system
student registration system
airline reservation system
electronic town meeting (or other group communication forum)
bulletin board/news system
electronic banking (e.g., ATM system)
group diary and appointment management system

Some faculty have students build software utilities similar to ones done in operating systems classes. They write: assembles, compilers, linker/loaders, or search/replace utilities. There are several drawbacks to such projects. They do not make a clear break between software engineering techniques and the methods they may use in operating systems classes. These projects still foster thought of development being the development of a program rather than a system with complex user interaction.

To overcome the later difficulty, it is important to develop systems with user interaction. There are several user application systems that can be developed. Some of these systems require extensive domain knowledge. Projects like this include:

economic analysis systems
library tracking systems
graduate tracking systems
Departmental calendar (Jones)
Group diary & appointments system (Sommerville, p 45 of instructor's guide)
Structure extractor (Jones) - extracts from source code: invocation hierarchy, DFD, object visibility chart, Nassi-Schneiderman chart, etc. (See Sommerville p 46 of instructor's guide)
CS1 Karel-Like Robot

Program comparer; plagiarism detector
Room scheduler
Football Rating system
Automobile Rental System
Fire and security alarm monitoring system (Sommerville, p 43 of IG)
Police vehicle command and control system (Sommerville, p 42 of IG)
software metric tool e.g., static structure analysis for a collection of modules
source-code management system (change control, reporting, data collection)
program visualizer (structure, etc.)
satellite tracking program
tax return calculation
hypertext authoring system
spreadsheet calculator
office automation system
expert system software
computer opponent for game playing (e.g., tic-tac-toe, othello)
simple flight simulator
simulate evolution (game of life)
revenue projector for concert administration
document concordance (generating index)
personal calendar
household budget
simulate a scientific, pocket calculator
newspaper typesetter
scoring athletic events
simple diagram editor (any type of diagrams)
interactive, symbolic manipulation of polynomials
computer animation, animated presentation system
diagram editor for electric power distribution systems
police vehicle command and control system
overhead projector transparency preparer

Sources of project ideas are listed in the software engineering bibliography. Texts that are especially useful in this regard are the books by: Blum, Booch, Frakes, Lamb, von Mayrhauser, Mynatt, Rakos, Rumbaugh et al., Schach, and Sommerville.

f. Inverted Functional Matrix Team Organization

The inverted functional matrix organization and management is described in the attached paper which has been submitted to SIGCSE 95 for presentation and publication. The figures referred to in the paper appear elsewhere in this report.

The Inverted Functional Matrix - A New Approach to Project Intensive Software Engineering Courses

Donald Gotterbarn, Robert Riser
East Tennessee State University

Suzanne Smith
Converse College

1. Introduction

The inverted functional matrix provides a new approach to organizing and managing projects in software engineering courses. It provides realistic experience in the development of large software products. Kurtz [6] has identified three typical approaches to class project organization: multiple teams of students independently developing the same project, multiple teams developing different projects, or one class project divided into programming subtasks where each subtask is assigned to a team. However, these models do not simulate real-world development of large projects as accurately as the inverted functional matrix organization.

This article describes the details of the project organization, the team structure, the management and assessment issues, and the benefits of this approach beyond other models.

2. Inverted Functional Matrix Organization

The inverted functional matrix organization incorporates attributes of both the functional and matrix models identified by Fairley [4]. This approach involves a single significant class project where the software development process model is used as the basis for project team organization. Individual teams are responsible for different life cycle phases. The columns of the matrix are the development teams, and the rows are the students. This matrix organization is inverted because the students are distributed to multiple functional teams of a single project rather than to a single functional team which is then distributed across multiple projects.

The inverted functional matrix organization is independent of process model. It has been used for both structured development and object-oriented development in classes at East Tennessee State University.

2.1. Team Organization

The partitioning of the class project into teams reflects the division of any software development project into analysis, design, and implementation phases, and support functions. The teams which start in the analysis phase are the requirements team, the user interface team, and the test plan team. The teams which start in the design phase are the preliminary design team and the detailed design team. The code team and the testing team begin during implementation. The support teams, configuration management and tools, exist throughout the project. The Gantt chart (Figure 1) shows this relationship.

The entire class is organized to work on a single project, and all students serve on either multiple teams or an entire life cycle support team. Figure 2 depicts the inverted functional matrix organization for a class of size fifteen; this organization has been scaled to other class sizes. The user interface team operates in both the analysis and design phases. The support teams, configuration management and tools, operate in all phases. One member of each analysis and design team is designated to maintain their team's deliverables until the completion of the project.

The choice of teams to which a given student is assigned is significant both to product quality and to breadth of experience to be gained by the student. To protect project integrity, careful attention must be given to the allocation of students to teams. No student is assigned to two teams which are responsible for validating one another's work; for example, no one is assigned to both the coding and the testing team. Correctly organized, teams act as cross checks on each other during development. For example, the user interface team meets independently with the user, while the requirements team meets with the customer. During the requirements review, the user interface team can help validate the requirements. The other consideration in the assignment of students to teams is the breadth of development experience gained by individual students. Members of the support teams experience the entire life cycle while on a single team. All other students serve on an analysis team, a design team and an implementation team. Members of the requirements team are split between the preliminary design and detailed design teams as the transition is made from analysis to design. The test plan team is split similarly. The user interface team remains intact through the design phase. As the transition is made from design to the implementation phase, each of the teams is distributed among the code and testing teams. While no student serves on every team, each becomes knowledgeable of the functions and products of all teams through the review process described in Section 3.4.

2.2. Team Definition

In this section, the teams for the inverted functional matrix organization are described according team the purpose and tasks, deliverables, and, where applicable, tools used by a team. Interactions between teams, and between a team and the customer or a team and the user are also described.

The requirements team meets with the customer in order to elicit, analyze, and specify the requirements for the software system. They develop the analysis model. For example, if structured analysis is used, the analysis model includes a narrative description of the proposed system, a list of requirements (acceptance criteria), context diagram, leveled data flow diagrams (DFDs), data dictionary, and process specifications. A CASE tool is used to produce the context diagram, leveled DFDs, and data dictionary. Such tools provide traceability and verifiability between the diagrams and the data dictionary.

The user interface team produces all user documentation for the system, develops user interface requirements, and designs the user interface of the system. Their deliverables are the preliminary format of the user manual, the complete user manual, and the detail for all user interface (e.g., required menus, forms, screens, reports, commands). A prototyping tool or screen generator is useful here.

The test plan team designs the subsystem and system tests. They develop the test plan which includes the test schedule, order of integration, checklist of tests to run at each step of integration, and traceability of tests to the requirement(s) being testing. They also design black-box test data (i.e., test data based on the requirements).

The preliminary design team creates a preliminary software structure of the system based on the requirements produced by the requirements team. The preliminary design deliverables are the system components (including a description of each component's functionality), the architecture of these components, the interface between the components, and a traceability matrix which relates a component to the requirement(s) which it satisfies. CASE tools can be used to create the system architecture, specify component interfaces, and describe component functionality.

The detailed design team creates the algorithms to implement the system structure produced by the preliminary design team. Detailed design's deliverables include the algorithms for each component delineated in the system structure and a traceability matrix which relates each algorithm to a component in the preliminary design. The algorithms can be specified by notations such as Nassi-Shneidermann charts, pseudocode, or flow charts.

The code team produces source code for the algorithms created by the detailed design team and performs unit testing. The tested source code is the deliverable for the code team. The minimal tools required are a text editor and a compiler for the specified programming language; however, other useful tools include a language-sensitive editor and debugger.

The testing team implements the tests described in the test plan and executes these tests in order to verify and validate the software. The deliverables of the testing team are the white-box tests and test data (i.e., data which exercises the logic of the system) and the documented test results. Finally they conduct acceptance testing. Any CASE tools which help automate the testing process are beneficial to this team and the code team.

The tools team provides training and ongoing support for the tools and environments needed by the other teams. They produce instructional materials (e.g., written tutorials and lectures) on the tools and environments. These instructional guidelines provide the basics for using a tool and details on any advanced features of a tool to which a team needs access.

The configuration management (CM) team develops and implements a configuration management plan. This team is in existence throughout the project and is normally a one-person team. The deliverables are the configuration management plan and all documentation necessary for its implementation. The configuration management plan includes documentation standards, configuration item control, and change control.

Some of the interaction between teams is evident in the descriptions above. Throughout the development process, teams produce deliverables which are needed by other teams to accomplish their tasks. For example, the preliminary design team works directly from the deliverables produced by the requirements team, and the detailed design team works directly from the deliverables of the preliminary design team. Other interactions are also needed for the teams to accomplish their tasks. The user interface team communicates with the requirements team in order to define the user interface and develop the user manual. The test plan team also communicates with the requirements team in order to develop tests and test data based on the requirements. The configuration management team and the tools team communicate with all other teams throughout the development process.

The interaction between teams and the customer or the user is also critical for the success of a project. Teams which have extensive communication with the customer or user are the requirements team, user interface team, and preliminary design team. This communication takes the form of interviews and participation in reviews of the deliverables for these teams.

3. Managing the Teams

It is essential to carefully control both inter-team and intra-team activity if the inverted functional matrix organization is to be effective. Objectives in team management include disciplined system development and overcoming student tendencies to procrastinate and become easily distracted from a task. Practical management techniques for accomplishing these objectives include an effective project start-up, a software project management plan, preliminary and final in-class reviews of team deliverables, regular team status reports, and inch pebbles plans.

3.1. Project Start-Up

Assigning all students to teams that begin work at once on the project is important because it sets the tone for the whole project. It is equally important that these teams be coordinated so that each has specific tasks to be undertaken immediately. Thus, the configuration manager, the requirements team, the user interface team, the test plan team, and the tools team are given specific assignments.

The configuration manager is charged immediately with developing a first draft configuration management plan. The configuration management plan, developed by the configuration manager and presented to the class for review, must be in place prior to the development or submission of any other configuration items.

All teams are informed of their specific deliverables. The requirements team immediately begins the process of eliciting requirements from the customer. Similarly, the user interface team interacts with the user and the requirements team to begin establishment of user interface requirements and a format for the preliminary user manual. The test plan team begins work on a test plan shortly after requirements analysis is underway. The tools team develops training materials and documentation for project support. With this start-up strategy, all students are immediately involved in the project.

3.2 Software Project Management Plan

A software project management plan (SPMP) describes the sequence of activities needed to successfully develop a particular software product. It minimally includes all major milestones and their due dates, the dates when teams are scheduled to begin work, and the list of configuration items (i.e., items to be placed under configuration control). If more detail is desired, the SPMP can also include intermediate targets such as walkthroughs and inspections. Figure 3 is a SPMP used through preliminary design. To be effective, the plan should be handed out when the project is introduced. Between deliverables, the plan should be referenced during class and students asked how they are progressing and if they need any help meeting their due dates. Just as the SPMP guides the students through the project, it also helps the instructor in managing the project.

3.3 Configuration Management

The SPMP underscores the role and visibility of configuration management. The configuration management plan establishes the place of the configuration manager in the development process. Documents submitted for final review, as specified on the SPMP, are immediately placed under CM and, therefore, configuration control. The first document placed under configuration control is the configuration management plan. A minimum form of configuration control can be established by setting up a directory which holds items placed under configuration control and to which only the configuration manager has write access and other class members have read access.

A good configuration manager makes the whole development process easier. Both the instructor and the configuration manager must promote the configuration management plan. Active and visible support for the configuration manager from the instructor is also necessary. The instructor acts as the configuration control board in handling change requests. The turn-around time on change requests must be minimized to avoid the perception that CM impedes rather than expedites development.

3.4 Reviews

In-class reviews are used as quality assurance activities for deliverables. Each milestone on the SPMP has a preliminary and final review. Reviews help students manage their team's progress in producing deliverables. Preliminary reviews also give teams early insights into the deliverables they will be receiving.

Before any review is conducted, appropriate review techniques [3, 7, 10] are discussed. It is important to stress that the function of reviews is the improvement of the delivered product and that problems are identified but not solved during a review.

The pattern for reviews is that the students give a preliminary review presentation in which each team member takes an active role. The material to be reviewed is distributed prior to the review. For each type of review, the class is given a review preparation form. These forms indicate specific items which are to be addressed in the review by the presenting team and by the student reviewers. Figure 4 is a sample of this form. Each student is expected to have carefully read the review material and to act as a reviewer. The team responds to questions or comments from the reviewers. Each team appoints a secretary to record all issues raised during a review. After the preliminary review, the team revises their deliverable for a later final review. The final review is conducted in the same manner as the preliminary review. After the final review, the team makes any remaining adjustments, and the deliverable is put under configuration management.

Student evaluations have been very favorable to the review process. A difficulty for the instructor is assuming the distinct roles of instructor and customer during the review. The customer expertise should be in the problem domain and not in computing. His/her primary concern is the system requirements be satisfied. However, it is sometimes appropriate during a review to assume the role of instructor. Balancing these dual roles is difficult and sometimes causes confusion. Post-project student assessments indicate that when a review is not going well that students would like to see more of the instructor than the customer.

3.5 Team Status Reports

Having two reviews does not prevent a last minute preparation frenzy just before a review. One way to encourage regular directed activity is the use of team status reports. A team status report is submitted for each team meeting. It includes a meeting agenda related to the team's current activities. The report indicates the disposition for each agenda item, the members present and absent, and the location and time of the meeting.
The team status report helps students direct their own efforts and to have more effective meetings. The status of each task is recorded. Some tasks are recorded as resolved while others may be assigned to particular students for resolution or for further study. Completing the sum of the tasks on the agenda moves the project team closer to their goal. This management tool does not significantly increase the students' work and enables the instructor to monitor the progress of all teams without attending all team meetings. They also help the instructor guide a team's progress. The goal of this process is not to catch a team or team members doing something wrong, but to guide a team toward the successful completion of its part of the project.

One of the hardest things in managing project intensive courses is determining the appropriate degree of instructor management of the project. While an instructor must not dictate a solution, he/she must provide appropriate and timely direction and coaching. Steering of a project takes several forms, from a simple question like "Have you handled X yet?", to attending a team meeting, or to suggesting an alternate solution. Attending a team meeting should be done sparingly lest the students perceive this as a "negative form of monitoring".

3.6 Inch Pebbles Plan

The best management does not always guarantee on-time delivery. When a project encounters difficulty, the students may want to "slip the schedule" by changing the due dates on the SPMP. In such instances, an inch pebbles plan can help students to measure and control their progress. An inch pebbles plan breaks a milestone into a series of smaller tasks for reaching that milestone. Such micro-management is only used when absolutely necessary. When we have had to resort to an inch pebbles plan, it has received universal praise in post-project assessments. A sample inch pebbles plan is in Figure 5.

As deadlines near, even with an inch pebbles plan, a common tendency is to resort to undisciplined development and to ignore or relax configuration management. Support for CM becomes even more important at this point. Scaling back on the required functionality of the project is preferable to giving up on CM. Abandoning CM conveys the wrong lessons. One way to scale back is to design the inch pebbles so that the system can be developed incrementally and at any pebble a functioning system can still be delivered. For example, the plan in Figure 5 is for a system that passes source code through a series of tests or filters. Even if the students completed coding only one filter, they would still produce a working system for an acceptance test.

The management techniques described above are not difficult but require constant attention. The return for this investment is a more mature development process. It includes productive team meetings, effective coordination of different teams' efforts, a clear understanding by the entire class of what each team is doing, reduction in the effects of procrastination, and a delivered software product.

4. Project Assessment Techniques

Feedback on the project must be provided to students throughout the course. Providing timely feedback can be difficult for the instructor, particularly in a project-oriented course because of the time demands associated with project and team management. We recommend multiple assessment components. As discussed in Section 3.4, properly conducted reviews are one valuable means of accessing the product being reviewed and the progress of its development.

Another valuable source of feedback is the assessment of "first drafts" of deliverables (e.g., user manual, requirements list, analysis model). These assessments provide early opportunities to steer teams in the right direction. First drafts are reviewed and returned by the next class meeting along with a written or oral critique. In either case, a record is kept and used to assure that the suggestions and/or criticisms are addressed in subsequent drafts.

An alternative method of providing quick feedback with minimal grading time is holistic grading. Holistic grading, which views an item as an integrated whole, involves a perusal of the entire item and the assignment of a single grade based on its entirety. Some student work is prone to holistic grading; some is not. For example, the first-level data flow diagram of a structured analysis model might be graded holistically on the following 0 to 4 scale.

4 - (a) balanced with context diagram;
(b) items appropriately described in data dictionary;
(c) processes appropriate for this level;
(d) captures essence of required functionality;
(e) adheres to process and data flow naming conventions;
(f) adheres to notational standards.

3 - Meets "4" description except for items (e) and (f), or fails to meet one of items (a) through (d).

2 - Meets "4" description except for items (e) and (f), and/or fails to meet two of items (a) through (d).

1 - Meets "4" description except for items (e) and (f), and/or fails to meet three of items (a) through (d).

0 - Meets "4" description except for items (e) and (f), and/or fails to meet any of items (a) through (d).

The scale is distributed with the returned work which is marked with a 0, 1, 2, 3, or 4. It is also made clear that the instructor will provide additional explanation for the assessment if asked. This rarely occurs.

The detailed assessment of deliverables associated with major milestones is also critical. "Product assessments" of deliverables, including the complete final package, are conducted. For example, Figure 6 is an excerpt from the evaluation of a structured analysis model that included a narrative description of the proposed system, a requirements list, context diagram, data flow diagrams, and data dictionary.

Peer and self evaluations are used extensively in assessing individual contributions to team projects. Students are informed that individual project grades are based on three factors: team project grade (determined in the product assessment), peer review, and the instructor's perceptions of individual contributions. Peer evaluation forms solicit student assessment of the team's products and interactions (Figure 7a) and of the contributions of team members(Figure 7b). A peer review is also conducted following preliminary design. While students find peer review difficult, post-project assessments indicate that they are generally appreciative of the opportunity to contribute their views.

In addition to the integration of continuous assessment into all project activities, a closing assessment has proven very productive. Often final delivery of the course project coincides with the final class meeting. This scheduling does not afford time for feedback or reflection. Class time should be provided to appraising the strengths and weaknesses of the process used and the products developed. To focus students on the assessment discussions, they are required to complete an assessment instrument. "Lessons learned" discussions result in students learning to be constructively critical of their own work and realistic about their plans. The discussions include an analysis of possible product improvements.

5. Conclusion

Although the inverted functional matrix approach to project intensive software engineering courses requires considerable effort from both the instructor and the students, its benefits are numerous and outweigh the difficulties.

In this approach to class project organization, students experience every aspect of software development. They are assigned to teams throughout the life cycle and receive in-depth exposure to those phases of the life cycle. Additionally, through participation in formal reviews and inter-team communication, they receive an understanding of the problems and tasks of the other phases of the life cycle.

Because student success and project success are dependent on many forms of communication, students learn to appreciate and practice these forms of communication. A higher level of precision is required in inter-team communications because teams which must communicate directly with each other have no common students. Informal speaking experience includes the communication with the customer or user and the intra-team and inter-team communications. Written skills are emphasized in the extensive documentation required in this approach.

Students, having participated in the development of significant deliverables, gain in-depth experience in at least two software development areas. Additionally, they experience the positive effects of controlling disciplines on the development of large projects. The inverted functional matrix organization is a feasible approach to the organization of project-oriented software engineering courses and provides real-world project experience beyond that available in other project organizations.

Copies of all the forms mentioned are available via electronic mail from the authors.

References

[1] Doris Carver, "Comparison of Techniques in Project-Based Courses," SIGCSE Bulletin, 17:1, March 1985.

[2] James Collofello, "Monitoring and Evaluating Individual Team Members in a Software Engineering Course," SIGCSE Bulletin, 17:1, March 1985.

[3] Lionel Deimel, "Scenes of Software Inspections: Video Dramatizations for the Classroom," Carnegie Mellon Technical Report, CMU/SEI-91-EM-5, 1991.

[4] Richard Fairley, Software Engineering Concepts, McGraw Hill, New York, 1985.

[5] Manmahesh Kantipudi, et.al.,"Software Engineering Course Projects: Failures and Recommendations," in Lecture Notes in Computer Science, Springer Verlag, C. Sledge, ed., 1992.

[6] Barry Kurtz and Tom Puckett, "Implementing a Single Classwide Project in Software Engineering Using Ada Tasking for Synchronization and Communication," SIGCSE Bulletin, 22:1, 1990.

[7] Barbee Mynatt,Software Engineering with Student Project Guidance, Prentice Hall, New Jersey, 1990.

[8] M. Rettig, "Software Teams," Communications of the ACM, 33:10, October 1990.

[9] Ian Sommerville, Software Engineering, 4th edition, Addison-Wesley, Massachusetts, 1992.

[10] Gerald Weinberg and Daniel Freedman, Handbook of Walkthroughs, Inspections, and Technical Reviews, Little, Brown and Company, Boston, 1982.

Real-World Software Engineering

V Example Case

Below is the project management plan and the test plan for an extended student project. This system examined two samples of Pascal source code checking for inappropriate similarities. We have not included either a requirements document nor a design document because there are numerous satisfactory examples of them in the literature.

A. Configuration Management

This is a section of the configuration management plan. The material on proper Ada coding style is not included here. It is chapter two of SPC71.

```
--------------------------------------------------------------------
|    PROJECT:       Third Eye Project
|    FILE NAME:     CM_PLAN.DOC
|    DOCUMENT NAME: Configuration Management Plan
--------------------------------------------------------------------
|    PURPOSE:
|         This document describes the responsibilities of
|         Configuration Management.
--------------------------------------------------------------------

|    MODIFICATION HISTORY:
|         WHO:               REV:          DATE:
|         Kellie Price       1.2           8/3/93
|            * Added CM_INCHS.DOC to C.M. files list
|
|         Kellie Price       1.1           7/16/93
|            * Added detailed design requirements
|
|         Kellie Price       1.0           7/12/93
|            * Created initial revision of document.
--------------------------------------------------------------------
```

Computer and Information Sciences
Third Eye Project

Configuration Management Plan

Kellie Price

Table of Contents

1. PURPOSE

The Configuration Management Plan defines the Configuration Management (CM) policies which are to be used in the Third Eye Project. It also defines the responsibilities of the project configuration manager.

2. MANAGEMENT

2.1 CONFIGURATION MANAGER RESPONSIBILITIES

The first responsibility of the configuration manager is to develop and implement this Configuration Management Plan.

Throughout the project, the configuration manager will report directly to the customer. It is the configuration manager's responsibility to ensure that the project is implemented in a straight-forward and well-defined manner according to the customer's specifications and standards established by Configuration Management for this project.

2.2 ORGANIZATION

This project will be divided into 7 teams as follows: (Refer to CM_TEAMS.DOC for the specific team assignments)

NOTE: All of the documents required of each team below are listed in the file CM_DOCS.DOC.

2.2.1 REQUIREMENTS TEAM

The Requirements Team is responsible for communicating with the customer in order to determine and well-define the software system requirements. The documents required of the Requirements Team are:

* Narrative description of system
* List of requirements (acceptance criteria)
* Context Diagram
* A series of leveled Data Flow Diagrams
* Data Dictionary
* Process Specifications

2.2.2 USER MANUAL TEAM

The User Manual team is responsible for producing all user documentation for the system. The documents required of the User Manual Team are:

* Preliminary format of user manual
* User Manual

2.2.3 TEST PLAN TEAM

The Test Plan team is responsible for designing subsystem and system tests. The documents required of the Test Plan Team are:

* Test plan

2.2.4 PRELIMINARY DESIGN TEAM

The Preliminary Design team is responsible for creating a preliminary design structure of the system based on the software system requirements. The documents required of the Preliminary Design Team are:

* An Object Model:
    * Complete object diagram
    * Class dictionary
    * Object-Requirements traceability matrix
* Ada Specifications for each object class

2.2.5 DETAILED DESIGN TEAM

This team is responsible for creating algorithms to implement the system structure. The documents required of the Detailed Design Team are:

* Data Structure design
* Algorithm design
* Traceability Matrix

2.2.6 CODE & UNIT TEST TEAM

The Code & Unit Test team is responsible for producing source code for the algorithms produced by the Detailed Design Team, testing each module of the system separately, and integration of the modules to produce a working system. The documents required of the Code & Unit Test Team are:

* Source code

2.2.7 TESTING TEAM

The Testing team is responsible for implementing the tests in the test plan and using them to test the system. The documents required of the Testing Team are:

* Test data
* Documented test results

3. CONFIGURATION MANAGEMENT ACTIVITIES

3.1 C.M. REQUIREMENTS DOCUMENTS

The configuration manager has provided documentation to assist the teams in meeting the C.M. requirements. This documentation is in a series of files which are available on the project file server. The C.M. requirements defined in these files are as follows:

DESCRIPTION	FILENAME
* Documents required by C.M.	CM_DOCS.DOC
* Document header info	CM_HEADR.DOC
* Document naming conventions	CM_NAMES.DOC
* Document format & standards	CM_FORMT.DOC
* Change request form format	CM_CHREQ.DOC
* Configuration item request procedure	CM_CIREQ.DOC
* Configuration item access procedure	CM_ACESS.DOC
* Configuration item change process	CM_CHPRO.DOC
* Configuration item baseline process	CM_BASLN.DOC

3.2 C.M. CONTROL

The configuration manager will provide the teams and team members controlled access to their respective configuration items. In order to have access, however, the teams and/or team members must provide the configuration manager with a written request for any desired configuration items as defined in the file CM_CIREQ.DOC.

4. CONFIGURATION MANAGEMENT RECORDS

All BASELINED Configuration Items and documents will be maintained on the project file server in a directory structure as defined in the file CM_FILES.DOC.

4.1 C.M. FILES

All Configuration Management files (including the requirements files listed in section 3.1) are listed below:

DESCRIPTION	FILENAME
* Configuration item access procedure	CM_ACESS.DOC
* Configuration item baseline process	CM_BASLN.DOC
* Configuration item change process	CM_CHPRO.DOC
* Change request form format	CM_CHREQ.DOC
* Configuration item request procedure	CM_CIREQ.DOC
* Original customer request	CM_CRQST.DOC
* Documents required by C.M.	CM_DOCS.DOC
* C.M. file directory structure	CM_FILES.DOC
* Change request form	CM_FORM.DOC
* Document format & standards	CM_FORMT.DOC
* Document header info	CM_HEADR.DOC
* Project Inch Pebbles	CM_INCHS.DOC
* Document naming conventions	CM_NAMES.DOC
* Document page header	CM_PGHDR.DOC
* Configuration Management Plan	CM_PLAN.DOC
* Software Project Management Plan	CM_SPMP.DOC
* Project team organization	CM_TEAMS.DOC

APPENDIX

```
+------------------------------------------------------------------
|    PROJECT:        Third Eye Project
|    FILE NAME:      CM_CRQST.DOC
|    DOCUMENT NAME:  Customer Request
+------------------------------------------------------------------
|    PURPOSE:
|         This document is the actual customer request for the
|         software system
+------------------------------------------------------------------
|    MODIFICATION HISTORY:
|         WHO:                 REV:                 DATE:
|         Dr. Riser(customer) 1.0                  6/22/93
|              * Created initial revision of document
+------------------------------------------------------------------
```

CSCI-4910 SOFTWARE SYSTEMS DEVELOPMENT WORKSHOP
SUMMER 1993 - GOTTERBARN/RISER/SMITH

PROJECT 2 - CUSTOMER REQUEST

As Dean, I often have to make the final decision, or make recommendations to a university hearing committee, in academic misconduct cases involving suspected plagiarism of student programs in computer science courses. I would like a system that can compare student programs and determine, with a high degree of credibility, whether plagiarism has occurred. I need an analysis report that is clear and understandable and could be presented as evidence to a university hearing committee.

The program will be used by faculty to screen student programs for possible plagiarism and to provide an objective analysis to support or negate their subjective opinions.

```
+------------------------------------------------------------------
|    PROJECT:        Third Eye Project
|    FILE NAME:      CM_SPMP.DOC
|    DOCUMENT NAME: Software Project Management Plan
+------------------------------------------------------------------
|    PURPOSE:
|         This document defines the plan to be followed during
|         the production of this system.
+------------------------------------------------------------------
|    MODIFICATION HISTORY:
|         WHO:                     REV:           DATE:
|         Gotterbarn/Riser/Smith   1.0            6/22/93
|              * Created initial version of this document
+------------------------------------------------------------------
```

Software Project Management Plan
Extended Project Summer 1993

Date	CI Id	Description
6/22		**Customer Request presented.**
6/28		**Team assignments announced and roles defined.**
6/29		**Start Development of Configuration Management Plan(CMP), Preliminary Requirements(P_REQ), Preliminary Test Plan(P_TP), and Preliminary User's Manual (P_UM).**
7/6	CI-1	**CMP delivered and presented to teams.**
7/6	CI-2	**P_REQ delivered and presented to teams and customers.**
7/8	CI-3	**P_TP delivered and presented to teams.**
7/8	CI-4	**P_UM delivered and presented to teams.**
7/12		**Requirements review. Preliminary design begins.**
7/13	CI-5	Final revised requirements is delivered and baselined by CM.
7/19		**Preliminary design review.**
7/20	CI-6	Final Preliminary Design is delivered and baselined by CM.
	CI-7	Final test plan is delivered and baselined by CM.
	CI-8	Final User Manual is delivered and baselined by CM.
7/26		**Detailed Design Review.**
7/27	CI-9	**Detailed design delivered and baselined by CM.**
8/6		Completed code goes to integration testing
8/9	CI-10	Source and Object Code Delivered.
8/10		**System acceptance test is conducted and the complete system and all associated documents are delivered to the customer.**

Items in bold denote student presentations.
All CI's are due at the beginning of class on the specified dates!

```
+------------------------------------------------------------------
|   PROJECT:       Third Eye Project
|   FILE NAME:     CM_INCHS.DOC
|   DOCUMENT NAME: Third Eye Project Inch Pebbles
+------------------------------------------------------------------
|   PURPOSE:
|        This document divides the remaining project 'milestones'
|        into 'inch-pebbles' so that the deadlines will be easier
|        met and progress may be made on several different teams at
|        the same time.
+------------------------------------------------------------------
|   MODIFICATION HISTORY:
|        WHO:              REV:          DATE:
|        Gotterbarn/Riser  1.0           7/29/93
+------------------------------------------------------------------
```

INCH PEBBLES, July 29-August 10th

Date	TESTING	CODING	DETAILED DESIGN
4:00 PM 7/30			**Nassi-S Charts, Data Structure Dictionary to CM, TEST, CODE**
7/30-8/2	Develop Tests for FILE, DRIVER SUMMARY REPORT, FILTER 1	Write FILE, DRIVER, SUMMARY REPORT, MENU, FILTER 1, SOURCEPROGRAM-F1	Develop trace matrix, help screens, Summary Report Format
8/2	Run tests on delivered code and Build tests for FILTER2	**Deliver 7/30 unit tested code to CM, and TEST.** Code all FILTER2 applications	**Deliver 7/30 items to CM, CODE and TEST**
8/3	**Discrepancy report on 8/2 tests to CM** Write test for FILTERS 3 and 4	**Deliver FILTER 2 CODE to CM and TEST** Write code for FILTER 3 and FILTER 4. Respond to discrepancy reports.	
8/4	**Discrepancy report on 8/3 tests to CM** Write tests for FILTERS 5 & 6 and Help screens.	**Deliver FILTERS 3 & 4 to CM and TEST.** Write code for FILTERS 5 & 6. Write Help screens. Respond to discrepancy reports.	
8/5	**Discrepancy report on 8/4 tests to CM** Write tests for FILTERS 7.	**Deliver FILTERS 5 & 6 to CM and TEST.** Write code for FILTERS 7. Respond to discrepancy reports.	
8/6	**Discrepancy report on 8/5 tests to CM** Write tests for FILTER 8.	**Deliver FILTERS 7 to CM and TEST.** Write code for FILTERS 8. Respond to discrepancy reports.	
8/9	**Discrepancy report on 8/6 tests to CM** Write accept.test	**Deliver FILTERS 8 to CM and and TEST.** Respond to discrepancy reports.	
8/10	Conduct acceptance test. **All test data to CM**		

NOTES: **Deliverables are in bold and both electronic & hard copies** are provided. All delivered code is to be unit tested. Discrepancy reports (DRs) are the results of testing by test team on baselined code. The CM passes DRs to CODE. When DR is fixed, a CR and modified code is given to CM. CM **baselines new code and delivers it to testing.**

```
+-------------------------------------------------------------------
|    PROJECT:        Third Eye Project
|    FILE NAME:      CM_FILES.DOC
|    DOCUMENT NAME:  C.M. File Directory Structure
+-------------------------------------------------------------------
|    PURPOSE:
|         This document defines file directory structure which |
     will be used by configuration management to file |
configuration items.
+-------------------------------------------------------------------
|    MODIFICATION HISTORY:
|         WHO:                REV:           DATE:
|         Kellie Price        1.1            8/9/93
|            * Added Detail and Prelim subdirectories
|         Kellie Price        1.0            7/12/93
|            * Created initial revision of document
+----------------------------------------------------------------
```

The file directory structure is as follows:

```
GROUPS --> CFGMGMT---------------+----> DOCUMENT -+---->MGMTDOCS
                                 |    +---> REQMENTS
                                 |    +---> DESIGN -+--> PRELIM
                                 |                  +--> DETAIL
                                 |    +---> TESTDOCS
                                 |    +---> USERMNUL
                                 +----> SOURCECD
                                 +----> TESTDATA
```

```
+------------------------------------------------------------------
|   PROJECT:        Third Eye Project
|   FILE NAME:      CM_ACESS.DOC
|   DOCUMENT NAME:  Configuration Item Access Procedure
+------------------------------------------------------------------
|   PURPOSE:
|        This document describes the procedure to be followed
|        when accessing a configuration item.
+------------------------------------------------------------------
|   MODIFICATION HISTORY:
|        WHO:                REV:            DATE:
|        Kellie Price        1.0             7/12/93
|           * Created initial revision of document
+----------------------------------------------------------------
```

In order to access the configuration items under configuration management control, the following steps may be taken:

NOTE: The project files are located on the network which can be accessed in Gilbreath Lab 105.

NOTE: The project files are available for READ access ONLY.

1. Pull up the main menu on the network
2. Press the 'ESCAPE' key
3. You should now be at the U:\> prompt
4. Type LOGIN CSCISERV/SECMUSER at the U:\> prompt
5. Enter your password. The password is CMUSER.
6. You will now be at the F:\> prompt
7. Type CD GROUPS\CFGMGMT at the F:\> prompt
8. You are now in the CFGMGMT directory.
9. Type DIR for a list of the subdirectories.
10. Choose the subdirectory you wish to access.
11. When you are finished, you MUST LOGOUT! To logout you must:
    - --> Type CD\ (return to the F:\> prompt)
    - --> Type CD LOGIN
    - --> Type LOGOUT

PLEASE, DO NOT FORGET TO LOGOUT!!!!!!

```
+-------------------------------------------------------------------
|   PROJECT:         Third Eye Project
|   FILE NAME:       CM_DOCS.DOC
|   DOCUMENT NAME:  Configuration Management Items
+-------------------------------------------------------------------
|   PURPOSE:
|        This document defines the documents that will be placed
|        under configuration management control.
+-------------------------------------------------------------------
|   MODIFICATION HISTORY:
|        WHO:                REV:            DATE:
|        Kellie Price        1.1             7/16/93
|           * Added Detailed Design documents
|
|        Kellie Price        1.0             7/12/93
|           * Created initial revision of document
+----------------------------------------------------------------
```

The following documents will be placed under configuration management control:

Requirements Documents:
- Narrative description of system
- List of requirements (acceptance criteria)
- Context diagram
- A series of leveled Data Flow Diagrams
- Data Dictionary
- Process specifications

User Manual Documents:
- Preliminary format
- User Manual

Test Plan Documents:
- Test plan

Preliminary Design Documents:
- Object Model:
  - Complete object diagram
  - Class dictionary
  - Object-Requirements traceability matrix
- Ada specifications for each object class

Detailed Design Documents:
- Data Structure design
- Algorithm design
- Traceability Matrix

Code Documents:
- Source code

Testing Documents:
- Test data
- Documented test results

```
+------------------------------------------------------------------
|    PROJECT:          Third Eye Project
|    FILE NAME:        CM_HEADR.DOC
|    DOCUMENT NAME:    Document Header Definition
+------------------------------------------------------------------
|    PURPOSE:
|         This document defines the document header that will be
|         at the top of all documents placed under configuration
|         management control.
+------------------------------------------------------------------
|    MODIFICATION HISTORY:
|         WHO:                REV:            DATE:
|         Kellie Price        1.0             7/12/93
|            * Created initial revision of document
+---------------------------------------------------------------
```

The format for the header is as follows:

NOTE: The text enclosed in '< >' must be replaced with the appropriate information by the document author.

```
+------------------------------------------------------------------
|    PROJECT:          <project name here>
|    FILE NAME:        <file name here......defined in CM_NAMES.DOC>
|    DOCUMENT NAME:    <document name here..(document description)>
+------------------------------------------------------------------
|    PURPOSE:
|         <purpose of this document>
+------------------------------------------------------------------
|    MODIFICATION HISTORY:
|         WHO:                REV:            DATE:
|         <person's name>     <new rev.#>     <date revised>
|            * <description of modification made>
|
|         <person's name>     1.0             <initial date>
|            * Created initial revision of document
+---------------------------------------------------------------
```

```
+----------------------------------------------------------------
|   PROJECT:        Third Eye Project
|   FILE NAME:      CM_NAMES.DOC
|   DOCUMENT NAME:  Configuration Item Naming Conventions
+----------------------------------------------------------------
|   PURPOSE:
|        This document defines the naming conventions to be used
|        for all items placed under configuration management
|        control.
+----------------------------------------------------------------
|   MODIFICATION HISTORY:
|        WHO:                REV:          DATE:
|        Kellie Price        1.2           8/2/93
|           * Added .PMD extension for PMDRAW files
|        Kellie Price        1.1           7/22/93
|           * Added .OMT extension for OMTOOL files
|        Kellie Price        1.0           7/12/93
|           * Created initial revision of document
+-------------------------------------------------------------
```

The valid file name prefixes are as follows:

Prefix	Team name
CM_	Configuration Manager
RQ_	Requirements Team
UM_	User Manual Team
TP_	Test Plan Team
PD_	Preliminary Design Team
DD_	Detailed Design Team
SC_	Coding Team (Source Code)
TD_	Testing Team (Test Data)
TR_	Testing Team (Test Results)

The remaining 5 characters are entirely up to the documents author. It is his/her responsibility to ensure that names are not duplicated within a team.

The valid file extensions are as follows:

Extension	Type
.DOC	WordPerfect file
.TST	System test file
.DFD	Case tool files (all leveled DFD's)
.DBF	Case tool file (Data dictionary)
.NDX	Case tool file (Data dict. index)
.OMT	Case tool file (OMTOOL)
.PMD	Case tool file (PMDRAW)
.ADA	Main program (driver) source file
.ADS	Package specification source file
.ADB	Package body source file

NOTE: The case tool files will have no prefix due to the fact that the case tools name the files automatically.

```
+-------------------------------------------------------------
|    PROJECT:        Third Eye Project
|    FILE NAME:      CM_FORMT.DOC
|    DOCUMENT NAME:  Document Format & Standards
+-------------------------------------------------------------
|    PURPOSE:
|         This document defines the document format standards
+-------------------------------------------------------------
|    MODIFICATION HISTORY:
|         WHO:                REV:           DATE:
|         Kellie Price        1.1            7/16/93
|            * Added detailed design notation.
|
|         Kellie Price        1.0            7/12/93
|            * Created initial revision of document
+-------------------------------------------------------------
```

1. GENERAL INFORMATION

1.1 MEDIA
All documents (files) will be submitted to configuration management as WordPerfect files (with the exception of source code, test data, and case tool files) on a 3.5" diskette.

1.2 NAMING
All documents (files) will follow the naming conventions defined in CM_NAMES.DOC.

1.3 VERSIONS
All versions of documents will begin at 1.0. Subsequent versions will be assigned a number of 1.X, where X is the next sequential number. The version number is not changed until the document has been baselined and a significant change has been requested, approved, and made.

1.4 HEADERS
All documents will contain a document header as defined in CM_HEADR.DOC.

2. STYLE

2.1 DOCUMENTS
All documents should adhere to the document standards presented in class.

2.2 SOURCE CODE
All source code should adhere to the coding standards of Ada which can be found in the Appendix of the Configuration Management Plan.

2.3 PRELIMINARY DESIGN
The Object Model will consist of:
1) A complete object diagram using Rumbaugh

notation as presented in class,

2) A class dictionary entry using notation presented in class,

3) An Object-Requirements traceability matrix using notation presented in class. Ada specifications should adhere to the coding standards of Ada which can be found in the Appendix of the Configuration Management Plan.

2.4 DETAILED DESIGN

The Data Structure Design will consist of a Data dictionary with both object entries and attribute entries.
The Algorithm design will consist of Nassi-Shneiderman models for each operation. The traceability matrix will be made up of two things: 1) Data Structures are related to Object_Attribute, and 2) NS-Models are related to Object_Operations. The specific notation for these documents are according to that which was presented in class.

3. FORMAT

The following sections describe a format which will be used for all appropriate configuration items (such as the User Manual and the Test Plan). Reference should be made to the Configuration Management Plan as an example of this format.

3.1 TITLE PAGE

All documents will have a title page which will include the following:

* Project name
* Document name
* Team (or author's) name

3.2 TABLE OF CONTENTS

All documents will have a table of contents which will look like the following:

Table Of Contents

```
1.      MAJOR IDEA.............................1
  1.1   SUPPORTING IDEA...................1
  1.2   SUPPORTING IDEA...................2
2.      MAJOR IDEA.............................3
  2.1   SUPPORTING IDEA...................4
  2.2   SUPPORTING IDEA...................4
        2.2.1  SUPPORTING IDEA............5
```

3.3 PAGE HEADER

All documents will have a page header on each page of text (not including Title page, Table of Contents, Index, or Appendix) which will include the document name and the page number. It is available in the file CM_PGHDR.DOC. (Refer to CM_ACESS.DOC for details in accessing the document.) It's format is similar to the following:

```
Document Name                                          1
--------------------------------------------------------
```

3.4 SECTION NUMBERING AND HEADERS

All documents will have the same numbering scheme. This numbering scheme is like that in the Table of Contents example above. Section headers will be ALL CAPS and indented as in the example also.

3.5 INDEX,APPENDIX

All documents are not required to have an Index or an Appendix, but these are optional and should be used when appropriate.

3.6 FONT,JUSTIFICATION

All documents should use the default fonts and justification values of WordPerfect 5.1.

```
+-------------------------------------------------------------
|    PROJECT:        Third Eye Project
|    FILE NAME:      CM_CHREQ.DOC
|    DOCUMENT NAME:  Change Request Form Definition
+-------------------------------------------------------------
|    PURPOSE:
|         This document defines the format for the change request
|         form.
+-------------------------------------------------------------
|    MODIFICATION HISTORY:
|         WHO:                REV:           DATE:
|         Kellie Price        1.0            7/12/93
|            * Created initial revision of document
+----------------------------------------------------------
```

CHANGE REQUEST FORM

Date:

Project:

Configuration Item to be changed:

Change requestor:

Requested change:

Improvement or repair:

Change request #:

Review date:

Reviewer:

Affected components:

Requirements Modification?:

Priority:

Estimated time:

Will be implemented?:

Change description:

Comments:

SIGN OFF LIST

Change implementor:	Date:
Configuration Manager:	Date:
Customer:	Date:

```
+----------------------------------------------------------------
|    PROJECT:         Third Eye Project
|    FILE NAME:       CM_CHPRO.DOC
|    DOCUMENT NAME:   Configuration Item Change Process
+----------------------------------------------------------------
|    PURPOSE:
|         This document describes the process to be followed when
|         changing a configuration item.
+----------------------------------------------------------------
|    MODIFICATION HISTORY:
|         WHO:                REV:           DATE:
|         Kellie Price        1.0            7/12/93
|            * Created initial revision of document
+--------------------------------------------------------------
```

1. When a change to the system or a configuration item is desired, a change request form must be submitted to the configuration manager.

   1.1 The change request form will be available in the file CM_FORM.DOC .Refer to CM_ACESS.DOC for details in accessing the documents.

   1.2 The change request form must follow the format described in the file CM_CHREQ.DOC.

   1.3 The change request form must be filled out as completely as possible by the person requesting the change.

2. The change request is reviewed by the configuration manager and appropriate project team members.

   2.1 A decision will be made as to whether or not the change is necessary and/or feasible. Requests in by 4 P.M. will be returned by 10 A.M. the following day.

3. The change is made and reviewed by the configuration manager and appropriate team members.

4. If the change modifies any customer requirements, the customer must also agree to it.

5. The change request form must be signed by the appropriate project team members.

```
+----------------------------------------------------------------------
|    PROJECT:         Third Eye Project
|    FILE NAME:       CM_CIREQ.DOC
|    DOCUMENT NAME:   Configuration Item Request Process
+----------------------------------------------------------------------
|    PURPOSE:
|         This document describes the process to be followed when
|         requesting a configuration item from configuration
|         management for modification.
+----------------------------------------------------------------------
|    MODIFICATION HISTORY:
|         WHO:                REV:           DATE:
|         Kellie Price        1.0            7/12/93
|            * Created initial revision of document
+--------------------------------------------------------------------
```

1. A written request is made to the configuration manager in the form of an approved change request. The request should include the following:

   1.1 The name(s) of the desired file(s) and the change request form number(s) if applicable.

   1.2 The name of the person responsible to see that the modified document(s) is re-submitted to the configuration manager.

   1.3 A BLANK FORMATTED 3.5" diskette.

   1.4 Any additional information the configuration manager may need.

NOTE: Once a baselined configuration item has been checked out, it will be locked and will not be available for additional changes until the updated version has been re-submitted and approved.

```
+----------------------------------------------------------------
|   PROJECT:         Third Eye Project
|   FILE NAME:       CM_BASLN.DOC
|   DOCUMENT NAME:   Baseline Process Description
+----------------------------------------------------------------
|   PURPOSE:
|        This document describes the process which will take
|        place when a configuration item is baselined.
+----------------------------------------------------------------
|   MODIFICATION HISTORY:
|        WHO:                REV:           DATE:
|        Kellie Price        1.0            7/12/93
|           * Created initial revision of document
+-------------------------------------------------------------
```

When a baseline or release is to be made, the following steps must be performed:

1. All appropriate configuration items will be submitted. The following conditions are required of each document:

   1.1 The document must be one of the documents listed in the file CM_DOCS.DOC.

   1.2 The document must represent the most up-to-date version in the configuration management files.

2. In the event that a baselined Configuration Item is changed, the Configuration Manager will notify the customer, if necessary, and each project team. Any necessary explanations will be provided at that time regarding the nature of the change.

TEST PLAN

This is part of the test plan for the plagarism project. The system did multiple comparisons on two Pascal programs. Each type of comparison was called a FILTER. There is a complete set of test documentation for a filter, but since the documents are similar for each filter, not every test is included.

```
+-----------------------------------------------------------------
|    PROJECT:       Third Eye Project
|    FILE NAME:     TP_TSTPL.DOC
|    DOCUMENT NAME: Test Plan
+-----------------------------------------------------------------
|    PURPOSE:
|         This document describes the testing strategy for the
|         Third Eye Project.
+-----------------------------------------------------------------
|    MODIFICATION HISTORY:
|         WHO:                 REV:           DATE:
|         Woodrow Beverley     1.2            8/08/93
|              * Changed expected results for name too long in
|              * test A3 from rejected to truncated. Also
|              * corrected the input data in these cases to
|              * exceed 32 characters. Added new case of (file
|              * already exists) to test A3.
|              *
|              * Changed test case of CONST type of FILE to SET.
|              *
|              * Corrected test data required section in tests
|              * B2b and B2c.
|              *
|              * Corrected test header in B2d.
|
|         WHO:                 REV:           DATE:
|         Woodrow Beverley     1.1            8/03/93
|              * 1) Changed all filter percentage test to have a
|              *    tolerance of + or - 4%
|              * 2) Deleted the expected physical line count and
|              *    Pascal line count from all test matrices.
|              * 3) Added Pascal line count and physical line
|                   data for filter one.
|
|         WHO:                 REV:           DATE:
|         Woodrow Beverley     1.0            7/28/93
|              * Created initial revision of document
+-----------------------------------------------------------------
```

Computer and Information Sciences
Third Eye Project

Test Plan

Woodrow Beverley
Mitch Moses
Drew Picklesimer

Table of Contents

1. INTRODUCTION

The Test Plan is presented in sections 3 and 4 of this document. Section 3 describes the methods to be used for the testing process. Section 4 contains the test procedures to be executed. These procedures are derived from the requirements specification document for the Third Eye Plagiarism Detection System. Blank copies of the forms presented in section 3 will be included in the Appendix.

2. REFERENCED DOCUMENTS

Client Request Version 1.0.
Configuration Management Plan Version 1.0.
Process Specifications Version 1.0.
Project Narrative Version 1.0.
Requirements Specification Version 1.0.

3. TEST METHODOLOGY

The following paragraphs will describe the items to be considered in the planning of the tests for the Third Eye Plagiarism Detection System.

3.1 TEST GROUP INVOLVEMENT

The Test Group will perform the integration tests, report any test failures to the Code Team, and ensure that all problems have been fixed at the end of the project. Also, the Test Group will be responsible for demonstrating the acceptance test to the customer.

3.2 REQUIREMENTS TRACEABILITY

The methodology for showing traceability of the requirements to the tests will be accomplished through a Test/Requirements Traceability Matrix shown on the following page. Each requirement will be represented in the matrix by its number in the requirements list. Each requirement will then be matched with a test or group of tests from the integration test list (see section 3.4).

The methods for verifying the requirements are as follows:

1) test and analysis of test results and
2) demonstration of the system.

Test/Requirements Testability Matrix

Requirement #	Tests
Functional	
1	A1
2	A1
3	B1a - B8d, F
4	A2, B1a - B8d
5	D
6	D
7	B1a - B8d
8	B1a - B8d
9	B1c,B2f,B3b,B4c,B5f,B6b,B7c,B8c,E
10	A3
11	A3
12	A3
13	A3
14	A3
15	A3
16	A3
17	A3
18	B1a - B8d
19	B1a - B8d, E
20	B1b
21	B1a
22	B1a
23	B1b
24	B1a
25	B1a
26	B1a - B1d
27	B1c
28	B2e
29	B2a
30	B2b
31	B2d
32	B2c
33	B2a - B2g
34	B2f
35	B3a - B3c
36	B3a - B3c
37	B3b
38	B4a - B5g
39	B4a - B5g
40	B4d, B5g
41	B4b
42	B4a
43	B4a
44	B4a - B4d
45	B4b, B5a
46	B4c

Requirement #	Test(s)
Functional	
47	B5a - B5g
48	B5a
49	B5b
50	B5d
51	B5c
52	B5a - B5g
53	B5a, B6a
54	B5f
55	B6a
56	B6b
57	B7a, B7d
58	B7b, B7d
59	B7a - B7d
60	B7c
61	B8a - B8d
62	B8a - B8d
63	B8c
Non-functional	
3	C

3.3 TEST SCHEDULE

The planning schedule shown below outlines the plan for completing the Test Plan, developing procedures, and test execution.

Test Schedule

Start Date	Complete Date	Activity
June 28	July 20	Complete Test Plan
July 6	July 20	Develop Procedures
July 26	August 5	Generate Test Data
August 6	August 10	Complete Test / Bug Fix Cycles

3.4 TESTS TO BE PERFORMED

The test list shown below shows test categories, any successful prerequisites needed by the test and the order in which the tests will be executed.

Tests to be Performed

Integration Tests	Order of Tests	Successful Prerequisite
A. Prompts		
1. Pascal source programs	1	--
2. Exit Test	2	A1
3. Report Information		
* Report name	4	B1a
* Course name		
* Professor's name		
* Students' names		
B. Filters		
1. Filter 1		
a. Physical lines / comments	3	A1
b. Constructs	5	
* Assignments		
* Procedure calls		
* Simple IF statements		
* Compound IF statements		
* Simple IF / ELSE statements		
* Compound If / ELSE statements		
* CASE statements		
* REPEAT UNTIL statements		
* WHILE statements		
* FOR statements		
c. Percentage	6	A1
* Two identical programs 100%		
* 79% test		
* 80% test		
* 10% test		
d. Torture	7	A1
* Comments and code on same line		
* All code commented out		
* Comment syntax in literal string		
* Pascal declarations		

Integration Tests	Order of Tests	Successful Prerequisite
2. Filter 2		B1
a. Global VARiable declarations	8	
* Array types		
* Boolean types		
* Char types		
* File types		
* Integer types		
* Real types		
* String types		
* User defined types		
b. Global CONSTant declarations	9	
* Array types		
* Boolean types		
* Char types		
* File types		
* Integer types		
* Real types		
* String types		
* User defined types		
c. Global TYPE declarations	10	
* Record types		
* User defined		
d. Global function/Procedure declarations	11	
e. Total number of global declarations	12	
f. Percentage	13	
* Two identical programs 100%		
* 79% test		
* 80% test		
* 10% test		
* 0% test		
g. Torture	14	
* Comments and code on same line		
* All code commented out		
* Multiple variables separated by commas		
* Functions declared in procedures		
* Procedures declared in procedures		

Integration Tests	Order of Tests	Successful Prerequisite
3. Filter 3		B2
a. Function/Procedure interfaces	15	
* Functions (Same number of parameters, same types, same order)		
* Procedures (Same number of parameters, same types, same order)		
* Mix of Functions/Procedures		
* Same number of parameters different types		
* Same number of parameters same type different order		
* Same number of parameters same type different parameters VARed		
b. Percentage	16	
* Two identical programs 100%		
* 79% test		
* 80% test		
* 10% test		
* 0% test		
c. Torture	17	
* Functions in procedures		
* Procedures in procedures		
* No functions or procedures		
* 99 procedures		
4. Filter 4		B3
a. Physical lines / comments	18	
b. Constructs	19	
* Assignments		
* Procedure calls		
* Simple If statements		
* Compound If statements		
* Simple If / Else statements		
* Compound If / Else statements		
* Case statements		
* Repeat Until statements		
* While statements		
* For statements		
c. Percentage	20	
* Two identical programs 100%		
* 79% test		
* 80% test		
* 10% test		
* 0% test		
d. Torture	21	
* Comments and code on same line		
* Empty procedure (Begin End;)		
* Comment syntax in literal string		
* Run repeatedly on two identical functions		

Integration Tests	Order of Tests	Successful Prerequisite
5. Filter 5		B4
a. VARiable declarations	22	
* Array types		
* Boolean types		
* Char types		
* File types		
* Integer types		
* Real types		
* String types		
* User defined types		
b. CONSTant declarations	23	
* Array types		
* Boolean types		
* Char types		
* User defined types		
* File types		
* Integer types		
* Real types		
* String types		
c. TYPE declarations	24	
* Record types		
* User defined		
d. Function/Procedure declarations	25	
e. Total number of global declarations	26	
f. Percentage	27	
* Two identical programs 100%		
* 79% test		
* 80% test		
* 10% test		
* 0% test		
g. Torture	28	
* Comments and code on same line		
* All code commented out		
* Multiple variables separated by commas		
* Run repeatedly on two identical functions		

Integration Tests	Order of Tests	Successful Prerequisite
6. Filter 6		B5
a. Keyword and control statements	29	
* Assignments		
* BEGIN/END pairs		
* ASSIGN		
* RESET		
* REWRITE		
* READ		
* READLN		
* WRITE		
* WRITELN		
* IF statements		
* CASE statements		
* REPEAT UNTIL statements		
* WHILE statements		
* FOR statements		
b. Percentage	30	
* Two identical programs 100%		
* 79% test		
* 80% test		
* 10% test		
* 0% test		
7. Filter 7		B6
a. Identifier names	31	
* Array types		
* Boolean types		
* Char types		
* File types		
* Integer types		
* Real types		
* String types		
* User defined types		
b. Functions/Procedures	32	
c. Percentage	33	
* Two identical programs 100%		
* 79% test		
* 80% test		
* 10% test		
* 0% test		
d. Torture	34	
* Same names different case		

Integration Tests	Order of Tests	Successful Prerequisite
8. Filter 8		B7
a. Functions	35	
b. Procedures	36	
c. Percentage	37	
* Two identical programs 100%		
* 79% test		
* 80% test		
* 10% test		
* 0% test		
d. Torture	38	
* Call functions declared in procedures		
* Call procedures declared in procedures		
* Procedure/Function calls in procedures		
* Same named function in both sources, but local in one and global in the other		
C. 30 seconds per response	41	B8
D. Help test	39	B8
* Main menu		
* Filter 1		
* Filter 2		
* Filter 3		
* Filter 4		
* Filter 5		
* Filter 6		
* Filter 7		
* Filter 8		
E. Total percentage test	40	B8
F. Menu Key torture test	42	

3.5 TEST PROCEDURES / RESULTS FORMS

A Test Procedure Form will be used to describe test procedures. A Test Results Form will be used to describe the process for recording test results. The forms to be used are shown in Figures 3.5-1 and 3.5-2.

NOTE: The text enclosed in '< >' will be replaced with the appropriate information for each test procedure.

TEST PROCEDURE FORM

Test: < Name of test (example: B1a) >

Test Version: < Version of test procedure >

Description: < Describes what the test covers >

Requirements: < These are numbers of the requirements from the Requirements Specification the test covers >

Prerequisites: < The names of any test that must be run before running this test >

Test Data Required: < DOS file names of Pascal source programs needed as test data by test >

Test Steps: < These are the actual test steps that the tester will follow when performing the test. The steps will also contain the expected result and a place for the tester to record the actual result of each step >

Figure 3.5-1

TEST RESULTS FORM

Test: < Name of test (example: B1a) >

Test Version: < Version of test procedure >

Executed By: < Name of tester >

Date Test Executed: < Date test was executed >

Version Number Tested: < Version of software tested, be sure to include beta number (example: 1.0 Beta 1 >

Test Results Passed __ Failed __ < Usa a X to mark the correct result >

Problems Identified: < Describe any problems found >

Test Steps: < These are the actual test steps that the tester followed while performing the test. The steps also contain the expected result and the actual result as recorded by the tester for each step >

Figure 3.5-2

3.6 TEST CHECKLIST FORM

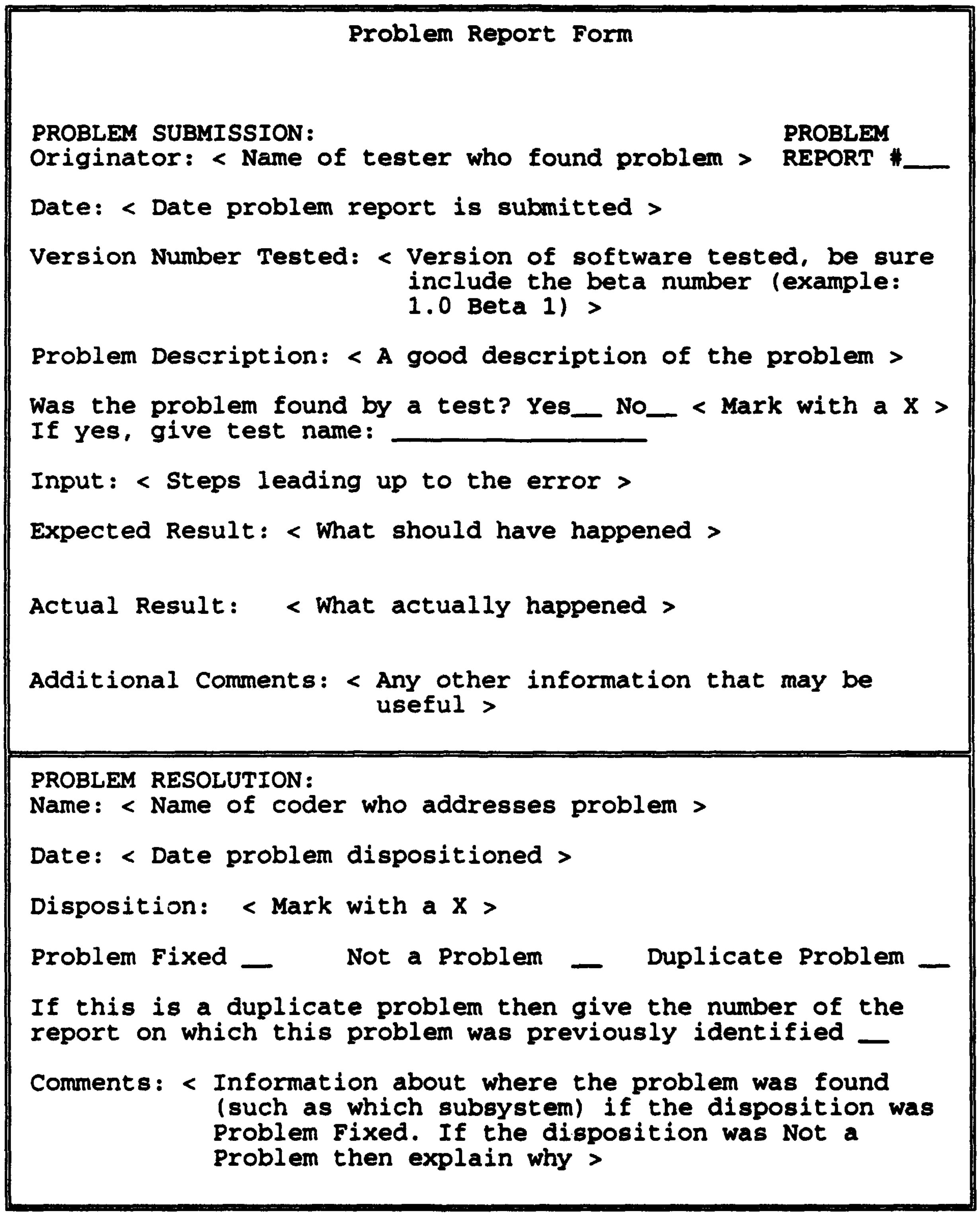
A test checklist will be maintained for each test cycle. The checklist will be used to track the status of all tests that have been executed during a test cycle. An example of a test checklist is shown in Figure 3.6-1. Also, a complete checklist form can be found in the Appendix.

Third Eye Plagiarism Detection System Ver 1.0 Beta 1.0					
Test Name	Description	Test Ver	Run By	Problem Report Numbers	Result P=Pass F=Fail N=Not Run
A1	Prompt for Pascal source programs				
A2	Exit Test				
...	...				

Figure 3.6-1

3.7 PROBLEM REPORT FORM

A Problem Report Form will be used to notify the Code Team of any problems found in the product during the test cycle. The Test Team will use the top half of the form and the Code Team will use the bottom half of the form. Each Problem Report will be assigned a unique number by a designated member of the Test Team. This designated member will also be responsible for tracking Problem Reports to ensure that all Problem Reports have been closed at the end of the project. A Problem Report will be considered closed after it has been dispositioned by the Code Team and verified by the Test Team. In order to aid in the tracking of the Problem Reports a Problem Report Tracking Form will be used. A Problem Report Form is shown in Figure 3.7-1 and a Problem Report Tracking Form is shown in Figure 3.7-2. Also, blank forms of the two forms can be found in the Appendix.



Problem Report Form

PROBLEM SUBMISSION:
Originator: < Name of tester who found problem >

PROBLEM REPORT #___

Date: < Date problem report is submitted >

Version Number Tested: < Version of software tested, be sure include the beta number (example: 1.0 Beta 1) >

Problem Description: < A good description of the problem >

Was the problem found by a test? Yes__ No__ < Mark with a X >
If yes, give test name: ________________

Input: < Steps leading up to the error >

Expected Result: < What should have happened >

Actual Result: < What actually happened >

Additional Comments: < Any other information that may be useful >

PROBLEM RESOLUTION:
Name: < Name of coder who addresses problem >

Date: < Date problem dispositioned >

Disposition: < Mark with a X >

Problem Fixed __ Not a Problem __ Duplicate Problem __

If this is a duplicate problem then give the number of the report on which this problem was previously identified __

Comments: < Information about where the problem was found (such as which subsystem) if the disposition was Problem Fixed. If the disposition was Not a Problem then explain why >

Figure 3.7-1

Problem Report Tracking Form					
Report Number	Ver Tested	Date Reported	Date Returned	Disposition: Fixed, Not a Problem, Duplicate	Closed
1					
2					
3					
4					
5					
6					
7					
8 ...					

Figure 3.7-2

3.8 SEQUENCE OF TEST EXECUTION FORM

Since the sequence that tests are executed can have an affect on weather a test fails or not a Sequence Of Test Execution Form will be maintained by each tester each test cycle. The form is shown in Figure 3.8-1 and a blank copy of this form can be found in the Appendix.

Sequence Of Test Execution Form

Tester Name ______________________

Version Tested ___________________

Test Order	Test Name	Date	Time	Comments
1				
2				
3				
4				
5				
6				
7				
8 ...				

Figure 3.8-1

4. TEST PROCEDURES

The following pages contain the actual test procedures that are to be executed by the Test Team.

TEST PROCEDURE FORM

Test: A1 (Pascal source programs)

Test Version: 1.0

Description: This test is performed to check the validity of two source program names.

Requirements: 1, 2

Prerequisites: None

Test Data Required: TD_1A.TST, TD_1B.TST

Test Steps:

1. Verify system asks for program name for first program.
2. Enter test data 1 through 5 from matrix for first Pascal name.

Note: In order to continue with the test, the system must accept the valid file name for first program in test data matrix number 5.

3. Verify system asks for program name for second program.
4. Enter test data 6 through 11 from matrix for second Pascal name.
5. Exit system.

TEST DATA MATRIX (A1)

Data Number and Description	Input	Expected Result	Actual Result	Pass/ Fail	Comments
1 (checks if name is too long)	Hellomynameistester.tst	Rejected			
2 (checks if no name entered)	Press <ENTER> key	Rejected			
3 (checks if bad characters entered)	Bad*?/.tst	Rejected			
4 (checks if two periods entered)	Td_1a..tst	Rejected			
5 (valid file name in lower case)	Td_1a.tst	Accepted			
6 (checks if name is too long)	Hellomynameistester.tst	Rejected			
7 (checks if no name entered)	Press <ENTER> key	Rejected			
8 (checks if bad characters entered)	Bad*?/.tst	Rejected			
9 (checks if two periods entered)	Td_1b..tst	Rejected			
10 (checks if name same as first)	TD_1A.TST	Rejected			Customer says this is OK do not run.
11 (valid file name in CAPS)	TD_1B.TST	Accepted			

TEST PROCEDURE FORM

Test: A2 (Exit test)

Test Version: 1.0

Description: This test is performed to check whether one can exit the system before filter one is run.

Requirements: 4

Prerequisites: A1

Test Data Required: None

Test Steps:

1. Verify system will allow you to exit prior to invoking filter one.

TEST PROCEDURE FORM

Test: A3 (Report information)

Test Version: 2.0

Description: This test is performed to check the validity of the data for the report file to be generated.

Requirements: 10 through 17

Prerequisites: B1a

Test Data Required: TD_1A.TST and TD_1B.TST

Test Steps:

Note: In order to continue with the test, the system must accept the valid data in test data matrix, numbers 5, 9, 13, and 17.

1. Enter TD_1A.TST for first program.

2. Enter TD_1B.TST for second program.

3. Run filter one.

4. Exit system.

5. Verify system asks for report name.
   Enter test data 1 through 5 from matrix for report name.

6. Verify system asks for course name.
   Enter test data 6 through 9 from matrix for course name.

7. Verify system asks for Professor's name.
   Enter test data 10 and 13 from matrix for Professor name.

8. Verify system asks for Student's name.
   Enter test data 14 through 17 for first Student's name.
   Enter test data 18 through 21 for second Student's name.

9. Verify report information in report file is same as information given in 9, 13, 17, and 21.

TEST DATA MATRIX (A3)

Data Number and Description	Input	Expected Result	Actual Result	Pass/ Fail	Comments
1 (checks if report name is too long)	Hellomynameistester.rpt	Truncated			
2 (checks if no name entered)	Press <ENTER> key	Rejected			
3 (checks if bad characters entered)	Bad*?/.rpt	Rejected			
4 (checks if two periods entered)	Td_a3..rpt	Rejected			
5 (checks if file already exists)	td_a1.tst	Rejected			
6 (valid report name in lower case)	Td_a3.rpt	Accepted			
7 (checks if course name is too long)	This name is much too long for course name	Truncated			
8 (checks if no name entered)	Press <ENTER> key	Rejected			
9 (checks if bad characters entered)	Bad*?characters	Rejected			Customer says not a problem do not run.
10 (valid course name entered)	Software engineering	Accepted			
11 (professor's name too long)	Donald Gotterbarns name is much too long	Truncated			
12 (checks if no name entered)	Press <ENTER> key	Rejected			
13 (checks if bad characters entered)	Bad*?characters	Rejected			Customer says not a problem do not run.
14 (valid professor's name)	Donald Gotterbarn	Accepted			
15 (Student one's name too long)	This name is much too long for student.	Truncated			
16 (checks if no name entered)	Press <ENTER> key	Rejected			
17 (checks if bad characters entered)	Bad*?characters	Rejected			
18 (valid student's name)	Student Name	Accepted			
19 (Student two's name too long)	This name is much too long for student.	Truncated			
20 (checks if no name entered)	Press <ENTER> key	Rejected			Customer says not a problem do not run.
21 (checks if bad characters entered)	Bad*?characters	Rejected			
22 (valid student's name)	John Doe	Accepted			

TEST PROCEDURE FORM

Test: B3c (Torture)

Test Version: 1.0

Description: This test is a stress test for filter three.

Requirements: 3, 4, 7, 8, 18, 19, 35, 36

Prerequisites: B2

Test Data Required: TD_52A.TST through TD_53B.TST

Test Steps: (Tests are to be iterative)

For each iteration of tests, do one through seven.

1. Enter program names given in the test data matrix.
2. Run filter one.
3. Run filter two.
4. Run filter three and record the results in test data matrix.
5. Exit after filter three.
6. Give report name as TD_B3C.RPT.
7. Verify report information in report file is same as actual results.

TEST DATA MATRIX (B3c)

Input Files	Expected Output						Actual Output						Pass/ fail	Comments
	List of functions/ procedures for file A			List of functions/ procedures for file B			List of functions/ procedures for file A			List of functions/ procedures for file B				
	Name	VA R	Not VAR	Name	VA R	Not VAR	Name	VA R	Not VAR	Name	VA R	Not VAR		
TD_52A.TST TD_52B.TST (Functions in Procedures)	Procedure1 Procedure2 Procedure3 Procedure4 Procedure5	0 0 0 1 2	0 1 2 3 4	Proced1 Proced2 Proced3 Proced4 Proced5	0 0 0 1 2	0 1 2 3 4								
TD_53A.TST TD_53B.TST (Procedures in Procedures)	Procedure1 Procedure2 Procedure3 Procedure4 Procedure5	0 0 0 1 2	0 1 2 3 4	Proced1 Proced2 Proced3 Proced4 Proced5	0 0 0 1 2	0 1 2 3 4								
TD_88A.TST TD_88B.TST (No functions or procedures)														
TD_89A.TST TD_89B.TST (99 procedures)	Procedure1 thru Procedure99	0 0 0	0 0 0	Proced1 thru Proced99	0 0 0	0 0 0								

TEST PROCEDURE FORM

Test: B5a (VARiable declarations)

Test Version: 1.0

Description: This test is performed to check all the different Pascal variable types. The test checks to see if the number of Pascal types are counted correctly in the VARiable section of selected procedures and functions. The test will check all of the following Pascal types: Array, Boolean, Char, File, Integer, Real, String, and enumerated (user defined types).

Requirements: 3, 4, 7, 8, 18, 19,47, 48, 52, 53

Prerequisites: B4

Test Data Required: TD_61A.TST and TD_61B.TST

Test Steps: (Tests are to be iterative)

1. Enter program names TD_61A.TST and TD_61B.TST.
2. Run filter one.
3. Run filter two.
4. Run filter three.
5. Run filter four. Choose any procedure names.

For each iteration of tests, do six and seven.

6. Run filter five.
7. Enter procedure names from test data matrix and record the results in test data matrix.
8. Exit after filter five.
9. Give report name as TD_B5A.RPT.
10. Verify report information in report file is same as actual results.

Note: Output for first and second program represented in form of first, second (ex. 10, 10).

TEST DATA MATRIX (B5a)

Function names From A, From B (Variable types defined in Procedure)	Expected Output (Number of VAR)	Actual Output	Pass/ fail	Comments
Procedure1, Proced1 Array types	5, 5			
Procedure2, Proced2 Boolean types	5, 5			
Procedure3, Proced3 Char types	5, 5			
Procedure4, Proced4 Enumerated types (User defined)	5, 5			
Procedure5, Proced5 File types	5, 5			
Procedure6, Proced6 Integer types	5, 5			
Procedure7, Proced7 Real types	5, 5			
Procedure8, Proced8 String types	5, 5			

TEST PROCEDURE FORM

Test: D (Help test)

Test Version: 1.0

Description: This test verifies that the user will receive context sensitive at each menu level.

Requirements: 5, 6

Prerequisites: B8

Test Data Required: None

Test Steps:

1. Start system and request HELP.
2. Verify help screen appears and is meaningful for this level.
3. Exit out of HELP.
4. Verify system returns you to prior screen before help request.
5. Record results in test matrix.
6. Enter program names given in the test data matrix.
7. Run filter one.
8. Request HELP.
9. Verify help screen appears and is meaningful for this level.
10. Exit out of HELP.
11. Verify system returns you to prior screen before help request.
12. Record results in test matrix.
13. Run filter two.
14. Request HELP.
15. Verify help screen appears and is meaningful for this level.
16. Exit out of HELP.
17. Verify system returns you to prior screen before help request.
18. Record results in test matrix.

19. Run filter three.
20. Request HELP.
21. Verify help screen appears and is meaningful for this level.
22. Exit out of HELP.
23. Verify system returns you to prior screen before help request.
24. Record results in test matrix.
25. Pick any procedure from first program and any procedure from second program and run filter four.
26. Request HELP.
27. Verify help screen appears and is meaningful for this level.
28. Exit out of HELP.
29. Verify system returns you to prior screen before help request.
30. Record results in test matrix.
31. Pick any procedure from first program and any procedure from second program and run filter five.
32. Request HELP.
33. Verify help screen appears and is meaningful for this level.
34. Exit out of HELP.
35. Verify system returns you to prior screen before help request.
36. Record results in test matrix.
37. Run filter six.
38. Request HELP.
39. Verify help screen appears and is meaningful for this level.
40. Exit out of HELP.
41. Verify system returns you to prior screen before help request.
42. Record results in test matrix.
43. Run filter seven.

44. Request HELP.

45. Verify help screen appears and is meaningful for this level.

46. Exit out of HELP.

47. Verify system returns you to prior screen before help request.

48. Record results in test matrix.

49. Run filter eight.

50. Request HELP.

51. Verify help screen appears and is meaningful for this level.

52. Exit out of HELP.

53. Verify system returns you to prior screen before help request.

54. Record results in test matrix.

55. Exit after filter eight.

TEST DATA MATRIX (D)

Menu Level	Expected Output		Actual Output		Pass/ fail	Comments
	Help Info Ok	Returned form help OK	Help Info OK	Returned from help OK		
Top Level	YES	YES				
Filter one	YES	YES				
Filter two	YES	YES				
Filter three	YES	YES				
Filter four	YES	YES				
Filter five	YES	YES				
Filter six	YES	YES				
Filter seven	YES	YES				
Filter eight	YES	YES				

LAB NUMBER: 011

TOPIC(S) FOR LAB:
Presentation of customer request for extended project.

INSTRUCTIONAL OBJECTIVE(S):
1. Introduce class to the customer for project 2 (the extended project).
2. Familiarize class with project 2.

ASSOCIATED LECTURE NUMBER:
Lecture 013

SET UP, WARM-UP:

The requirements for the your first (small) projects were better defined than is often the case. Quite often, the customer has nothing more than a general idea of the type of system he/she wants. In such cases, the extraction of requirements is even more critical and challenging.

PROCEDURE:

1. Introduce the "real" customer to the class in order to explain the system that he/she would like developed. If the instructor is acting as the customer, it is important that he/she role play and resist the urge to switch roles during the lab. The customer request goes something like the following:

   "As Dean, I often have to make the final decision, or make recommendations to a university hearing committee, in academic misconduct cases involving suspected plagiarism of student programs in computer science courses. I would like a system that can compare student programs and determine, with a high degree of credibility, whether plagiarism has occurred. I need an analysis report that is clear and understandable and could be presented as evidence to a university hearing committee.

   The program will be used by faculty to screen student programs for possible plagiarism and to provide an objective analysis to support or negate their subjective opinions."

   The customer responds to questions.

2. Distinguish between the customer and user. For the extended project, the dean is the customer and user(s) will be faculty members and the dean.

3. Give teams the remainder of the time to work on the project.

ASSOCIATED HANDOUTS:

LECTURE NUMBER:012

TOPIC(S) FOR LECTURE:

The importance of controlling disciplines in software development
Configuration management
Ways to implement configuration management

INSTRUCTIONAL OBJECTIVE(S):

1. Recognize the role of configuration management over the entire life cycle.
2. Develop and evaluate a configuration management plan.

SET UP, WARM-UP:
(How involve learner: recall, review, relate)

L11OH4

We have just talked about maintenance. Maintenance is change to software that occurs after a system is developed. As we have seen, some errors are introduced into the software during the maintenance process. The development of software is a continuous process of change and affords developers a continuous opportunity to introduce errors into the system. Some would consider this opportunity for the introduction of error unacceptable. We cannot alter the nature of the development process, but if we manage and control the process of change we can restrict the opportunities for introducing error.

(Learning Label- Today we are going to learn ...)

In software engineering, the principles for controlling and managing change are called configuration management. Today we are going to look at the principles of configuration management and ways in which configuration management can be implemented.

CONTENTS:

1. Motivate the need for configuration management(CM) by discussing the simultaneous update problem and version control. CM is not just an issue about software. You have revised your small project requirements list several times. Ask the students what changes were made to the requirements list and whether they all were certain that all would have been approved by the customer? Did you check with the customer?

   a. Multiple people working on a large project can have different understandings of what the system is supposed to do or may make small changes which do not work well with other parts of the system. Using the test plan L9OH6 remind the student that the test planner made a change which required the KoFF system to track expired cards. Was this information communicated to the designers, coders or even to the customer? What is the likelihood of this change made by the test planner ever getting implemented? What can be done to assure that these kinds of changes are acceptable and will get implemented? There is a need for control management and communication of change.

b. There are multiple sources and reasons for change requests. These occur throughout the development process as well as after system development. Talk about change requests as desired improvements in the system. As the customer better understands the system he/she sees new and improved ways that the system could be developed. Changes also come from the developers' improved understanding of the requirements, and changes in the environment while a system is being developed. This can lead to chaos unless carefully managed.

c. Sometimes systems are developed in different versions, e.g., DOS 2.2 - DOS 6.0. Each version of each system has to be tracked and maintained. Versions are not always sequentially developed, as was DOS 4, DOS 5, and DOS 6. Sometimes multiple versions of the same system are developed concurrently to fit on different hardware platforms, e.g., UNIX for DEC and IBM can be developed at the same time. The requirements for these systems are different, and one must track and maintain multiple versions of the "same" product.

d. Talk about baselining as a technique for limiting or controlling this chaos. How is baselining done? Be sure to emphasize that this involves formal review and agreement by all concerned parties. Once an item is baselined it is under change control and can only be changed by formal change control procedures. What is it that gets baselined. Proposed changes to baselines are called Change Requests(CR).

2. Methods of CM require a plan, a well defined process, and a manager to carry out the plan.

a. Ask the class what things they need to keep constant to develop a system. List these on the board. Discuss them as Configuration Items (CI). Display overhead of standard configuration items L12OH1. Work through each item talking about those items which are new to them. Be sure to emphasize that any change requires approval and communication of the change as well as updating the affected documents. Another function of CM is to maintain consistency between the documentation of a system and the system itself.

b. CM is complex and requires a plan to be sure it is executed. Display overhead L12OH2. Go over the contents of the IEEE CM Plan. Briefly go over the management issues, such as how configuration management relates to other organizations. Discuss overhead item 2.d which includes naming conventions for components and how CRs will get processed. Distribute handout L12HD1 as an example of a portion of a student-produced CM plan.

c. To maintain control, baselined configurations items are sometimes placed in a special electronic library. Permission to change or modify CIs is gained through a CR approval process. CRs are generally approved by groups called <u>Configuration Control Boards (CCB)</u>. Discuss some standards used to decide the approval of CRs; e.g, functional need, cost versus benefit analysis, impact on other modules, politics (the president of the company "just wants it").

d. There are several virtues of CM which include reducing the number of errors generated, minimizing the use of storage, giving visibility to system development progress each time a new CI is baselined and reducing the time and effort costs associated with uncontrolled change.

<u>PROCEDURE</u>:

<u>teaching method and media</u>:

At this point in the course students have likely experienced uncontrolled changes within their small projects. Some of these have also likely caused problems. Their own "war stories" can serve to enhance their interest and appreciation of the necessity for configuration management. The primary teaching technique consists of using lecture and overheads with frequent reference to problems they have encountered in their small project teams.

<u>vocabulary introduced</u>:

configuration management (CM)
configuration item (CI)
baseline
discrepancies versus changes
configuration control board (CCB)
change request (CR)

INSTRUCTIONAL MATERIALS:

overheads:

L12OH1	Configuration items
L12OH2	IEEE Model for a configuration management plan

handouts:

L12HD1	Student configuration management plan

RELATED LEARNING ACTIVITIES:
(labs and exercises)

Lab 010 - Feedback on CI-5, test plans, and test cases.Small project team preparation for team acceptance test presentations

READING ASSIGNMENTS:

Sommerville Chapter 29 (pp. 551-564)
Mynatt Chapter 8 (pp. 336-340)

RELATED READINGS:

Ghezzi Chapter 7 (pp. 403-408)
Pressman Chapter 21 (pp. 693-708)
Schach Chapter 4 (pp. 87-93)
James Tomayko, "Support Materials for Software Configuration Management," Support materials, SEI_SM_4_1.0
IEEE Standard for Software Configuration Management Plans, IEEE Std 828-1983

Configuration Items

Requirements Documents
- Client Request
- Requirements List
- Analysis Documents
- Revision History
- Revision requests and approvals

Design Documents
- Preliminary Design Documents
- Preliminary Design Review Documents
- Detailed Design Documents
- Detailed Design Review Documents
- Revision History
- Revision requests and approvals

Code Documents
- Source code modules
- Object code modules
- Compiler used
- System build plan

Other Documents
- Test Plan
- Test Cases
- Test Results
- User Manual
- Referenced Documents

IEEE Model for a CONFIGURATION MANAGEMENT PLAN

1. Introduction
   a. Purpose
   b. Scope
   c. Definitions and acronyms
   d. References

2. Management
   a. Organization
   b. Configuration management responsibilities
   c. Interface control
   d. Implementation of plan
   e. Applicable policies, directives and procedures

3. Configuration management activities
   a. Configuration identification
   b. configuration control
   c. Configuration status accounting
   d. Audits and reviews

4. Tools, techniques, and methodologies

5. Supplies Control

6. Records collection and retention

Configuration Management Plan

```
+---------------------------------------------------------
|    PROJECT:               Third Eye Project
|    FILE NAME:        CM_PLAN.DOC
|    DOCUMENT NAME:         Configuration Management Plan
+---------------------------------------------------------
|    PURPOSE:
|         This document describes the responsibilities of
|         Configuration Management.
+---------------------------------------------------------
|    MODIFICATION HISTORY:
|         WHO:                             REV:             DATE:
|         Kellie Price
|           * Created initial revision of document.
+---------------------------------------------------------
```

Computer and Information Sciences
Third Eye Project

Configuration Management Plan

Kellie Price

Table of Contents

1. PURPOSE

The Configuration Management Plan defines the Configuration Management **(CM)** policies which are to be used in the Third Eye Project. It also defines the responsibilities of the project configuration manager.

---

2. MANAGEMENT

2.1 CONFIGURATION MANAGER RESPONSIBILITIES

The first responsibility of the configuration manager is to develop and implement this Configuration Management Plan.

Throughout the project, the configuration manager will report directly to the customer. It is the configuration manager's responsibility to ensure that the project is implemented in a straight-forward and well-defined manner according to the customer's specifications and standards established by Configuration Management for this project.

2.2 ORGANIZATION

This project will be divided into 7 teams as follows:
(Refer to CM_TEAMS.DOC for the specific team assignments)

NOTE: All of the documents required of each team below are listed in the file CM_DOCS.DOC.

2.2.1 REQUIREMENTS TEAM

The Requirements Team is responsible for communicating with the customer in order to determine and well-define the software system requirements. The documents required of the Requirements Team are:

- Narrative description of system
- List of requirements (acceptance criteria)
- Context Diagram
- A series of leveled Data Flow Diagrams
- Data Dictionary
- Process Specifications

2.2.2 USER MANUAL TEAM

The User Manual team is responsible for producing all user documentation for the system. The documents required of the User Manual Team are:

* Preliminary format of user manual
* User Manual

2.2.3 TEST PLAN TEAM

The Test Plan team is responsible for designing subsystem and system tests. The documents required of the Test Plan Team are:

* Test plan

2.2.4 PRELIMINARY DESIGN TEAM

The Preliminary Design team is responsible for creating a preliminary design structure of the system based on the software system requirements. The documents required of the Preliminary Design Team are:

* An Object Model:
    * Complete object diagram
    * Class dictionary
    * Object-Requirements traceability matrix
* Ada Specifications for each object class

2.2.5 DETAILED DESIGN TEAM

This team is responsible for creating algorithms to implement the system structure. The documents required of the Detailed Design Team are:

* Data Structure Design using a data structure dictionary

* Algorithm Design using Nassi-Shneiderman models

* An object attributes and object operations traceability matrix

2.2.6 CODE & UNIT TEST TEAM

The Code & Unit Test team is responsible for producing source code for the algorithms produced by the Detailed Design Team, integration of the modules to produce a

working system. The documents required of the Code & Unit Test Team are:

* Source code

2.2.7 TESTING TEAM

The Testing team is responsible for implementing the tests in the test plan and using them to test the system. The documents required of the Testing Team are:

* Test data
* Documented test results

3. CONFIGURATION MANAGEMENT ACTIVITIES

3.1 C.M. REQUIREMENTS DOCUMENTS

The configuration manager has provided documentation to assist the teams in meeting the C.M. requirements. This documentation is in a series of files which are available on the project file server. The C.M. requirements defined in these files are as follows:

DESCRIPTION	FILENAME
* Documents required by C.M.	CM_DOCS.DOC
* Document header info	CM_HEADR.DOC
* Document naming conventions	CM_NAMES.DOC
* Document format & standards	CM_FORMT.DOC
* Change request form format	CM_CHREQ.DOC
* Configuration item request procedure	CM_CIREQ.DOC
* Configuration item access procedure	CM_ACESS.DOC
* Configuration item change process	CM_CHPRO.DOC
* Configuration item baseline process	CM_BASLN.DOC

3.2 C.M. CONTROL

The configuration manager will provide the teams and team members controlled access to their respective configuration items. In order to have access, however, the teams and/or team members must provide the configuration manager with a written request for any desired configuration items as defined in the file CM_CIREQ.DOC.

4. CONFIGURATION MANAGEMENT RECORDS

All BASELINED Configuration Items and documents will be maintained on the project file server in a directory structure as defined in the file CM_FILES.DOC.

4.1 C.M. FILES

All Configuration Management files (including the requirements files listed in section 3.1) are listed below:

DESCRIPTION	FILENAME
* Configuration item access procedure	CM_ACESS.DOC
* Configuration item baseline process	CM_BASLN.DOC
* Configuration item change process	CM_CHPRO.DOC
* Change request form format	CM_CHREQ.DOC
* Configuration item request procedure	CM_CIREQ.DOC
* Original customer request	CM_CRQST.DOC
* Documents required by C.M.	CM_DOCS.DOC
* C.M. file directory structure	CM_FILES.DOC
* Change request form	CM_FORM.DOC
* Document format & standards	CM_FORMT.DOC
* Document header info	CM_HEADR.DOC
* Document naming conventions	CM_NAMES.DOC
* Document page header	CM_PGHDR.DOC
* Configuration Management Plan	CM_PLAN.DOC
* Software Project Management Plan	CM_SPMP.DOC
* Project team organization	CM_TEAMS.DOC

LAB NUMBER: 012

TOPIC(S) FOR LAB:
Organization of extended project

INSTRUCTIONAL OBJECTIVE(S):
1. Provide project organization to be used for extended project.

ASSOCIATED LECTURE NUMBER:
Lecture 014

SET UP, WARM-UP:
You have already met the customer for the extended project when he/she presented the project request recently. Today we will discuss the team organization to be used for the extended project.

PROCEDURE:

1. HANDOUT - List of teams in extended project organization
Distribute and describe the role and responsibilities of each team.

    a. Explain that during this semester the project will be completed through preliminary design. Thus the teams that will be active during this semester are configuration management, requirements, user interface, test plan, and preliminary design (and tools if a tools team in included in the project organization).

    b. During the course of the project each student will serve on more than one team. In particular, each student will serve on a "high end" team (encompasses analysis through preliminary design) and a "low-end" team (encompasses detailed design through implementation).

    c. Communication will be more complex in this project than in your small projects. Both inter-team and intra-team communication will be necessary, as well as communication with the customer and users. For example, crucial interactions from the beginning will include:

    Configuration management with instructor and all other teams.

    Requirements team with customer, user, and other teams.

    User interface team with user and requirements team.

    Test plan team with requirements team and user interface.

    d. Four teams will begin work as soon as the team assignments are made. Configuration management will begin developing a CM plan, requirements will begin eliciting and specifying the requirements, user interface will begin on the format of the users manual and the "look and feel" of the user interface, test plan will begin developing the format of the test plan.

4. Invite any students with a preference for specific team assignments to let instructor know as soon as possible. Let students know that they cannot be guaranteed their preferences but that they will be considered.

ASSOCIATED HANDOUTS:

List of teams
Extended project team organization

EXTENDED PROJECT TEAMS

Configuration management

Requirements

Test plan

User Interface

Preliminary design

Detailed design

Code and unit test

Testing

Tools (optional)

Inverted Functional Matrix Organization Model

	Analysis		Design		Impl.			Support	
Team/ Student	Req	Tst Pl	User Int	Prel Dsgn	Det Dsgn	Code	Test	CM	Tools
1	*			*					
2	*			*					
3	*			*					
4	*			*					
5	*			*					
6	*			*					
7			*						
8			*						
9			*						
10			*						
11			*						
12		*		*					
13		*		*					
14		*		*					
15		*		*					
16		*		*					
17								*	
18								*	
19									*
20									*
21									*

NOTE: Preliminary design team will be subdivided into two or more teams when design begins.

LAB NUMBER: 013

TOPIC(S) FOR LAB:
Peer/self assessment - small projects
Acceptance reviews - small projects

INSTRUCTIONAL OBJECTIVE(S):
1. Provide opportunity for peer/self assessment of small project teams.
2. Simulate acceptance test/review for small projects.

ASSOCIATED LECTURE NUMBER:
Lecture 015

SET UP, WARM-UP:
As discussed in our first class meeting, we want to consider your opinions in assessing the team projects. We have prepared a peer/self evaluation instrument and will be administering it today, prior to your acceptance test presentations. Immediately following that we will conduct the acceptance test presentations.

PROCEDURE:
1. HANDOUT - Peer/self evaluation forms for small projects
   Ask each student to complete a peer/self evaluation form for his/her team. Stress that their responses are confidential and will be seen by the instructor only. While feedback will be provided to individuals, the confidentiality of responses will remain strictly confidential.

ASSOCIATED HANDOUTS:
Peer/self evaluation forms for small projects
Oral presentation evaluation form
Material to be reviewed has been provided to reviewers in advance.

KIOSK TEAM: PEER/SELF EVALUATION **NAME: ________________**

Your responses are confidential and will be seen only by the instructors. Be completely honest. Use back for additional comments.

1. Evaluate the performance of each team member, <u>including yourself</u> with respect to each of the following questions by indicating **SA** (strongly agree), **A** (agree), **D** (disagree), or **SD** (strongly disagree).

____ ____ ____ ____ ____ ____

He/she took a fair share of the responsibility and work.

He/she took a leadership role.

He/she kept aware of the project's problems and progress.

He/she is knowledgeable of the tools and techniques used.

He/she attended meetings and cooperated with rest of team.

He/she gave an honest effort and completed tasks on time.

I would choose to work with him/her on another project.

2. Complete Columns A and B for each team member, including yourself.

COLUMN A: Enter +, =, or - as follows.

+ means this person made a significant contribution to the team and should be given a bonus; their individual project grade should be higher than the team grade.

= means this person did their share; their individual project grade should be equal to the team grade.

- means this person's performance was less than adequate and his/her individual project grade should be lower than the team grade.

COLUMN B: Describe his/her major contributions.

TEAM MEMBER	(A) + = -	(B) MAJOR CONTRIBUTION(S)
1.		
2.		
3.		
4.		
5.		
6.		

3. For each item below, rate your team's performance and deliverables produced. UNS represents unsatisfactory and EXC represents excellent.

(a) Interaction with user in understanding/defining requirements

```
             BELOW                ABOVE
UNS          AVG        AVG       AVG        EXC
|-------|-------|------|------|------|------|------|------|
```

(b) Configuration Item 1 - narrative description (abstract) of project and requirements list.

```
             BELOW                ABOVE
UNS          AVG        AVG       AVG        EXC
|-------|-------|------|------|------|------|------|------|
```

(c) Configuration Item 2 - analysis documents: context diagram, leveled data flow diagrams, data dictionary.

```
             BELOW                ABOVE
UNS          AVG        AVG       AVG        EXC
|-------|-------|------|------|------|------|------|------|
```

(d) Configuration Item 3 - design documents: system architecture (structure chart and external description of modules and interfaces).

```
             BELOW                ABOVE
UNS          AVG        AVG       AVG        EXC
|-------|-------|------|------|------|------|------|------|
```

(e) Configuration Items 4 and 5 - test plan (classes of tests for each requirement); test scenarios (specific tests, input, expected output, etc.)

```
             BELOW                ABOVE
UNS          AVG        AVG       AVG        EXC
|-------|-------|------|------|------|------|------|------|
```

(f) Configuration Items 6 and 7 - documented source code; executable.

```
             BELOW                ABOVE
UNS          AVG        AVG       AVG        EXC
|-------|-------|------|------|------|------|------|------|
```

(g) Configuration Item 8 - documentation of testing; acceptance testing plans, documentation, etc.

```
               BELOW                   ABOVE
UNS             AVG        AVG          AVG         EXC
|-------|-------|------|------|------|------|------|------|
```

(h) Configuration Item 7 - Acceptance test review

```
               BELOW                   ABOVE
UNS             AVG        AVG          AVG         EXC
|-------|-------|------|------|------|------|------|------|
```

(i) Overall, rate the your team's performance for the entire project?

```
               BELOW                   ABOVE
UNS             AVG        AVG          AVG         EXC
|-------|-------|------|------|------|------|------|------|
```

(j) Overall, the tools team and the materials they have produced have been

```
NO            LITTLE                                          MUCH
HELP           HELP              OK             HELP          HELP
|---------|---------|--------|--------|--------|--------|--------|--------|
```

4. If you had to do this project again and were in charge of hiring personnel, in what order would you rehire the team? In other words, who would be the person on your team you would rehire first, second, third, etc.? (Be sure to include yourself).

1.

2.

3.

4.

5.

6.

<u>LAB NUMBER</u>: 014

<u>TOPIC(S) FOR LAB</u>:

Assessment of small projects.
Team assignments for extended projects.

<u>INSTRUCTIONAL OBJECTIVE(S)</u>:

1. Provide assessment of products produced by teams.
2. Provide assessment of individual team members.
3. Organize teams for extended project.
4. Provide software project management plan for extended project.

<u>ASSOCIATED LECTURE NUMBER</u>:

Lecture 016

<u>SET UP, WARM-UP</u>:

Recall that the course policies explained that there were three factors to be considered in your individual project grades: the product produced by the team, peer assessment of individual contributions to the team effort, and the instructor(s)' assessment of individual contributions. Today each of you will receive a copy of our assessment of your project with a team project grade and with an individual grade for your contribution to the project. We will then announce the team assignments for the extended project.

<u>PROCEDURE</u>:

1. Distribute individual assessment reports containing the product grade, the individual grade, and extended comments on the product.

2. Invite teams or individuals to make appointments to discuss the assessments. Stress that we will discuss the peer/self assessments with individuals but only in composite terms; confidentiality will be maintained.

3. Announce team assignments for extended project. Distribute team rosters.

4. HANDOUT - Software project management plan for extended project
Distribute and discuss the project management plan.

5. Teams are given the remaining time for organizational meetings.

<u>ASSOCIATED HANDOUTS:</u>

Small project team evaluations (samples attached)
Rosters for extended project teams
Software project management plan - extended project

PROJECT 1 EVALUATION: KIOSK VENDING MACHINE SYSTEM

TEAM MEMBER: ______________________

TEAM PRODUCT GRADE: ___ **TEAM MEMBER'S GRADE:** ___

COMMENTS ON DELIVERED PRODUCT

These comments pertain to the delivered software product and are not necessarily reflective of the time and/or effort expended.

The quality of the delivered document was first rate. There were, however, several problems which made the system less than perfect. Most of the comments below are items of clarification. Our major concerns with your product were about the structure chart, the coded system that was delivered, and the absence of interface descriptions.

There were several concerns about the data dictionary.

Physical money and amount representation still gave you some problems, e.g., **change** and **current amount** are defined the same, yet one represents physical money and the other is a representation of money. Refund has a similar problem.

Items needs iteration.

slot number- The iteration is for the number of integers in the slot number and not for the range of numbers. The numerals 1 to 32 are either integer + integer or integer with a lower bound of 1 for the numerals 0 to 9 and an upper bound of 2 for the numerals 10 to 32.

In a context diagram you should use nouns, e.g., dispense and cancel are not nouns.

We had some questions about the level 0 DFD. **Alarm status** is not accessed at all. Its function is not clear.

Data stores in a DFD should appear only on one level, but **running total data** and **stock store** each appear on two levels.

The structure charts created several problems.

There was no top level structure chart for your system. What is the **system** you intend to deliver to the customer? This is a major problem.

There should not be multiple information items on data couples.

The structure chart and the code should agree at least in terms of the number of inputs and outputs. This is not the case. See for example dispense items and validate selection. The source code and the design should agree. One should map the other, i.e., logically they should be in the same order even if the code breaks things down into smaller modules.

The process narratives were well done!

Interface descriptions were missing.

CODE The customer has no code! The customer was given a test scaffold which ties together several discrete modules to enable an acceptance test. Where is the system that the customer can run once the test is done?

TESTING There are no results recorded on any of the test result forms. The only difference from one test result form to the next is the requirements number on the form. There should have been some major concerns with some of the tests, for example the dispense test doesn't check to see if the quantity in a slot has been decreased after an item was dispensed. These forms indicate that version 1.0 was tested and yet the documents indicate that version 2.0 was submitted.

PROJECT 1 EVALUATION: FIRE AND SECURITY ALARM SYSTEM

TEAM MEMBER: ______________________

TEAM PRODUCT GRADE: ___ **TEAM MEMBER'S GRADE:** ___

COMMENTS ON DELIVERED PRODUCT

These comments pertain to the delivered software product and are not necessarily reflective of the time and/or effort expended.

OVERALL PACKAGING - Product gives appearance of having been thrown together at last minute, including some items being crossed out and others being penciled in.

NARRATIVE DESCRIPTION - In paragraph 7, "same span of time .... " is still vague to be tested.

REQUIREMENTS LIST

#9 and Footnote are inconsistent with one another.
#13, #15: indentation erratic.
#20: incident reports and frequency reports never fully defined.

CONTEXT DIAGRAM, DFDs, DATA DICTIONARY

Some items missing from data dictionary, including:

Notify
Incident report
Frequency report
Incidents
Signal to Fire Equipment
Signal to Warning Device
Signal to Lock/Unlock
Room Function
Data stores not defined in data dictionary.

Use of "Flag" in data dictionary is still awkward. It would be much more meaningful to do something like the following:

Hazard level = High | Normal
Type = Fire | Security
Alarm condition = In-service | Out-of-Service

In places, more meaningful names could be used; e.g., Type is too vague.

Context diagram and DFD are not balanced; leveling of DFD incomplete.

DFD is rough; far from form expected in finished product.

As shown in model, SetUpFile should be an external entity.

DESIGN DOCUMENTS

No external description of modules and interfaces submitted.

Various inconsistencies or omissions in structure chart; for example Get Info (page 2) doesn't return any information.

Module (and procedure) names should always be as descriptive as possible and consist of Verb and Object.

CODE

Design and code are inconsistent.

In places it is tough to distinguish between test modules and product modules; for example the OurTime procedure.

Programming standards were not followed in several aspects including identifier dictionaries, input and output descriptions, use of meaningful identifier names (for example, look at TimeCompare and TimeSubtract).

Inconsistent documentation blocks (for example, none on Init and CallProcedures).

Comments - look at II.7 of programming standards.

What is purpose of procedure CallProcedures?

TEST PLAN, TEST RESULTS

Test categories too broad; for example consider Section C (Alarm Responses) - these need to be broken down further to adequately test system.

Test procedure form: (a) all look like low-level unit tests; (b) all end with the generic statement "Verify test data is output to test result file." Should also worry about correctness of output.

Test results somewhat confusing and appear inadequate; not mappable to test procedures.

Software Project Management Plan: Extended Project

Week.Class	CI Id	Description
1a		Customer request presented.
2b		Team assignments announced and roles defined.
		Start development of configuration management plan (CMP), preliminary requirements (P_REQ), preliminary test plan (P_TP), and preliminary user's manual (P_UM).
4b	CI-1	CMP delivered and presentation to teams.
5a	CI-2	P_REQ delivered; presentation to teams and customer.
	CI-4	P_UM delivered; presentation to teams.
5b	CI-3	P_TP delivered; presentation to teams.
6a		Requirements review. Preliminary design begins.
6b	CI-5	Final revised requirements baselined.
8b		Preliminary design review.
9a		User manual review Test plan review
9b	CI-6 CI-7 CI-8	Final preliminary design delivered and baselined Final test plan delivered and baselined Final user manual delivered and baselined
14a		Detailed design review
14b	CI-9	Detailed design delivered and baselined
19a		Unit tested code goes to integration testing
21a	CI-10	Source and object code delivered
21b	CI-11	System acceptance test conducted and the complete system and all associated documents are packaged and delivered to the customer

NOTE: All presentation/review items are distributed to designated reviewers 24 hours prior to the presentation/review.

LAB NUMBER: 015

TOPIC(S) FOR LAB:
Initial user perspective of extended project

INSTRUCTIONAL OBJECTIVE(S):
1. Provide perspective of user of system to be developed for extended project.

ASSOCIATED LECTURE NUMBER:
Lecture 017

SET UP, WARM-UP:
You have already met the customer for the extended project when he/she presented the project request recently. Today a prospective user of the system is going to present his/her perspective and answer any questions you might have. Following your meeting with the user we will discuss the project organization to be used for the extended project.

PROCEDURE:

1. Remind the students of the difference between the customer and the user. Then introduce the user to give his/her initial view of the system and to answer any questions.

   NOTE: It is important to plan in advance requirements that are to be explicitly identified by the user during this discussion and requirements that will be mentioned only if elicited by student questions. For example, consider the plagiarism detection system described previously by the dean (customer). For this system, the items bolded below are to be explicitly communicated by the user; non-bolded items below will be communicated during requirements extraction process as appropriate questions arise.

   Input is two Pascal source programs, from CS-1 and CS-2 students.

   Want menu-driven interactive system.

   See system as a series of filters with a stop after each filter to show filter's results and for a Go/No-Go response from the user.

   Output report produced at each filter summarizing its results.

   First level (filter): # lines of code (Pascal statements)
   # lines of comments (between { })
   total # physical lines (eoln's)

   Second level (filter) - considers main (global) declarations
   # variables   \# variables by type

# user-defined types	# procedures declared
# constants	# functions declared

Third level (filter) - considers structure of procedure/function interfaces

parameters, type, order

Fourth level (filter) - Filter 1 applied to selected procedures/functions

Fifth level (filter) - Filter 2 applied to selected procedures/functions

Sixth level (filter) - considers statement types for main body only

Frequencies for assignments, reads & readlns, writes & writelns, IFs, WHILEs, FORs, REPEAT-UNTILs, CASEs, BEGIN-ENDs

Seventh level (filter) - considers name matching

Eighth level (filter) - adherence to standards (?)
- unusual structures (BEGIN-END problems, etc)

2. The user answers questions.

ASSOCIATED HANDOUTS:
Extended project team organization

LAB NUMBER: 016

TOPIC(S) FOR LAB:
Immediate tasks for configuration management, requirements, user interface, and test plan teams.

INSTRUCTIONAL OBJECTIVE(S):
1. Begin work on deliverables for extended project.

ASSOCIATED LECTURE NUMBER:
Lecture 018

SET UP, WARM-UP:
In recent labs the project organization has been explained and team assignments made. The customer and a user have given their perspectives on what is needed. Each of you has been assigned to a team. Today we're going to talk about immediate tasks and give you time to work on the project.

PROCEDURE:

1. For each team, discuss the work that needs immediate attention.

   a. Configuration management
      i Read Sommerville Chapter 29;
      ii Begin draft of CM plan

   b. Requirements - Begin requirements definition

   c. User interface
      i Read Mynatt Appendix B
      ii Begin developing user manual format
      iii Begin considering "look & feel" of system

   d. Test plan - Begin developing test plan format

   e. Tools (optional)
      i Begin identifying development tools to be used and begin developing expertise in their use
      ii Begin planning training and training materials in tools use

2. Stress the importance of communication between teams.

3. Provide each team with a suitable work area and give them this time to work on their immediate tasks. The instructor and the user should be available.

ASSOCIATED HANDOUTS:

LAB NUMBER: 017

TOPIC(S) FOR LAB:
Configuration management plan presentation/review

INSTRUCTIONAL OBJECTIVE(S):
1. Configuration manager presents configuration management plan for review.

ASSOCIATED LECTURE NUMBER:
Lecture 020

SET UP, WARM-UP:
Remind students that the purpose of these reviews are to improve the item under review. All participants (developers, customers, and other reviewers) have a common goal: to identify issues that need to be addressed.

Remind them as well of some of the guidelines for reviews that have been discussed in the past. Include: during the review we want to identify problems, not attempt to solve them; avoid the tendency to resist change or be defensive; remember this is a team effort.

Remind them to note any issues identified that require attention and to follow up on the issues list as soon as possible.

PROCEDURE:

1. Introduce the configuration manager and begin the review.

ASSOCIATED HANDOUTS:
Oral presentation evaluation form
Material to be reviewed has been provided to reviewers in advance.

LAB NUMBER: 018

TOPIC(S) FOR LAB:

Preliminary requirements presentation/review
Preliminary users manual presentation/review

INSTRUCTIONAL OBJECTIVE(S):

1. Requirements team presents preliminary requirements for review.
2. User interface team presents format of users manual for review.

ASSOCIATED LECTURE NUMBER:

Lecture 021

SET UP, WARM-UP:

Remind students that the purpose of these reviews are to improve the item under review. All participants (developers, customers, and other reviewers) have a common goal: to identify issues that need to be addressed.

Remind them as well of some of the guidelines for reviews that have been discussed in the past. Include: during the review we want to identify problems, not attempt to solve them; avoid the tendency to resist change or be defensive; remember this is a team effort.

Remind them to note any issues identified that require attention and to follow up on the issues list as soon as possible.

PROCEDURE:

1. Discuss briefly the need for each team to carefully review the work of other teams. This is particularly important to assure consistency between teams requirements, user interface, test plan, and configuration management.

2. Introduce the requirements team and begin the review.

3. Introduce the user interface team and begin the review.

ASSOCIATED HANDOUTS:

Oral presentation evaluation form (2)
Material to be reviewed has been provided to reviewers in advance.

LAB NUMBER: 019

TOPIC(S) FOR LAB:

Preliminary test plan presentation/review

INSTRUCTIONAL OBJECTIVE(S):

1. Test plan team presents preliminary test plan for review.

ASSOCIATED LECTURE NUMBER:

Lecture 022

SET UP, WARM-UP:

Remind students that the purpose of these reviews are to improve the item under review. All participants (developers, customers, and other reviewers) have a common goal: to identify issues that need to be addressed.

Remind them as well of some of the guidelines for reviews that have been discussed in the past. Include: during the review we want to identify problems, not attempt to solve them; avoid the tendency to resist change or be defensive; remember this is a team effort.

Remind them to note any issues identified that require attention and to follow up on the issues list as soon as possible.

PROCEDURE:

1. Discuss briefly the need for each team to carefully review the work of other teams. This is particularly important to assure consistency between teams requirements, user interface, test plan, and configuration management.

2. Introduce the test plan team and begin the review.

ASSOCIATED HANDOUTS:

Oral presentation evaluation form

Material to be reviewed has been provided to reviewers in advance.

LAB NUMBER: 020

TOPIC(S) FOR LAB:
Final requirements presentation/review

INSTRUCTIONAL OBJECTIVE(S):
1. Requirements presents final requirements for review.

ASSOCIATED LECTURE NUMBER:
Lecture 023

SET UP, WARM-UP:
The work of all of the teams has been carefully reviewed by all of you. This began with a review of the preliminary requirements followed by user interface and test plan reviews, respectively. During each review issues have arisen that needed to be addressed. Some issues identified involved inconsistencies in the way different teams were interpreting aspects of the system. We expect that reviewing other teams' work has caused each of you to examine your own work in an effort to resolve the inconsistencies.

By this time all issues should have been addressed and should be reflected in modifications to the appropriate documents. Likewise they should be reflected in today's requirements review.

As always, remind students to note any issues identified that require attention and to follow up on the issues list as soon as possible.

PROCEDURE:

1. Stress the importance of this review. The requirements will be baselined once issues uncovered in this review have been addressed. Point out the importance of this baseline to provide a stable base for design.

2. Introduce the requirements team and begin the review.

ASSOCIATED HANDOUTS:
Oral presentation evaluation form
Material to be reviewed has been provided to reviewers in advance.

LAB NUMBER: 020

TOPIC(S) FOR LAB:

Final requirements presentation/review

INSTRUCTIONAL OBJECTIVE(S):

1. Requirements presents final requirements for review.

ASSOCIATED LECTURE NUMBER:

Lecture 023

SET UP, WARM-UP:

The work of all of the teams has been carefully reviewed by all of you. This began with a review of the preliminary requirements followed by user interface and test plan reviews, respectively. During each review issues have arisen that needed to be addressed. Some issues identified involved inconsistencies in the way different teams were interpreting aspects of the system. We expect that reviewing other teams' work has caused each of you to examine your own work in an effort to resolve the inconsistencies.

By this time all issues should have been addressed and should be reflected in modifications to the appropriate documents. Likewise they should be reflected in today's requirements review.

As always, remind students to note any issues identified that require attention and to follow up on the issues list as soon as possible.

PROCEDURE:

1. Stress the importance of this review. The requirements will be baselined once issues uncovered in this review have been addressed. Point out the importance of this baseline to provide a stable base for design.

2. Introduce the requirements team and begin the review.

ASSOCIATED HANDOUTS:

Oral presentation evaluation form

Material to be reviewed has been provided to reviewers in advance.

LAB NUMBER: 021

TOPIC(S) FOR LAB:
Preliminary design

INSTRUCTIONAL OBJECTIVE(S):
1. Steer preliminary design.

At this point the extended project teams are approaching a critical juncture. The instructor needs to have a thorough understanding of where the teams are heading. This understanding, of course comes form a variety of sources: reviews, interactions with those involved in the project (student teams, customer, user), and "unofficial" comments from individual students on team progress. Generally a number of different approaches have been suggested by different students. In order to increase the chances for successful completion of the project, the instructor may need to steer the design. Issues the students may not be aware of include hardware constraints and language capabilities and constraints. Appropriate steering may involve guiding students towards a particular design approach and/or away from potential pitfalls.

ASSOCIATED LECTURE NUMBER:
Lecture 026

SET UP, WARM-UP:

Today we want the project teams to continue work on the project. There are several issues on which we need clarification and which we want to ask about as we visit with teams during the lab.

PROCEDURE:

1. Informally meet with individual teams or entire class, whichever is appropriate, and provide necessary steering.

ASSOCIATED HANDOUTS:

LAB NUMBER: 022

TOPIC(S) FOR LAB:
Ada laboratory environment

INSTRUCTIONAL OBJECTIVE(S):
1. Be acquainted with the Ada environment on the machines in our local lab.
2. Understand how to enter and compile a Ada source program in lab.

ASSOCIATED LECTURE NUMBER:
Lecture 027

SET UP, WARM-UP:

You will be implementing the extended project in Ada. Today we want to explain the Ada tools and environment that you will be using and let you begin experimenting with them.

NOTES:
1. Develop a short handout on your particular Ada environment. This should include instructions for accessing all tools and for editing, compiling, linking, and executing Ada programs.

2. If a tools team is used, they should be responsible for all or part of this laboratory.

PROCEDURE:

1. HANDOUT - Description of Ada laboratory environment
In the lab, walk through the handout with students.

   a. The Ada compiler is our lab is a line editor which is not the most user-friendly environment to work in; therefore, we have set up the lab so that Turbo Pascal editor is on every machine with the Ada compiler. You can enter your programs using the Turbo Pascal editor and then exit to compile the program using the Ada compiler.

   b. Before using the Ada compiler, you must set up the Ada environment on your diskette in the A drive. The command necessary to set of the Ada environment is **newlib** which only has to be done once for each diskette. This command will create an ADA.LIB program and an ADA.AUX directory on your diskette. This command creates a local library database that references the standard library database file provided by the compiler. A link from your diskette back to the Ada compiler in the C:\ADA directory is also established. All your work should be done on the A drive. The next commands are shown using the A drive prompt.

   c. In order to compile a program, the Ada compiler must be invoked with the source code file name. For this example, let's say that the program just created was in a file named "try.ada" and the name of the main program (the internal name of the program) was "try". To compile an Ada programming unit, type the following at the prompt:

```
A:\>ada try.ada
```

You must include the file extension when compiling. This command will indicate, in its screen output, the line number and type of error for any compile errors encountered in the program. The Turbo Pascal editor gives, as you move through a program, the current line number on the bottom left of the screen; these line numbers are in agreement with the line numbers given by the compiler on error messages.

All programming units must be compiled in the order of their dependency. The final programming unit to be compiled should be the main program which utilizes other programming units to perform a task.

d. If the main program required no corrections and recompilations, the next step is to invoke the Ada linker which produces an executable program by putting together the separate components of the program. To link the compiled program, type the following at the prompt:

```
A:\>bamp try
```

The name of the main program (in this case "try") is the only information needed by the linker. An executable program file (given the name try.exe) is produced by the linker.

e. To run the executable program, type the name of the file (without the extension) at the prompt:

```
A:\>try
```

2. To practice entering and compiling a program, use the program given in your Benjamin textbook on page 2. Pages 2-4 explain the actions and statements in this program.

3. Also included in the Ada environment on the machines in our local lab is an interactive Shareware Ada tutor. This is a self-paced introduction to Ada. [John Herro, The Interactive Ada-Tutor, Software Innovations Technology, 1083 Mandarin Drive N.E., Palm Bay FL. 32905-4706]

To access this tutorial software package, type the following sequence of commands starting at the DOS C drive:

```
C:\>cd ada
C:\ada>cd adatutor
C:\ada\adatutor>ada-tutr
```

ASSOCIATED HANDOUTS:

Description of Ada laboratory environment

Working in Ada Environment In Lab 104

For all these tasks, start at the DOS prompt (if working on OS/2 machine, go to DOS Full Screen). To prepare your diskette for compiling and running Ada programs:

1. Put formatted diskette in drive A.

2. Change directory to drive A.

3. Type "newlib" at the prompt.

   ```
   A:\>newlib
   ```

   This command will create an ADA.LIB program and an ADA.AUX directory on your diskette. This command creates a local library database that references the standard library database file provided by the compiler. A link from your diskette back to the Ada compiler in the C:\ADA directory is also established. This command has to be done only once for a diskette unless these items are deleted from the disk.

4. To type in a program, use the Turbo Pascal editor which can be accessed from the A: drive by simply typing "turbo" at the prompt.

   ```
   A:\>turbo
   ```

5. For this example, let's say that the program just created was in a file named "try.ada" and the name of the main program (the internal name of the program) was "try". To compile an Ada programming unit, type the following at the prompt:

   ```
   A:\>ada try.ada
   ```

   You must include the file extension when compiling. This command will indicate, in its screen output, the line number and type of error for any compile errors encountered in the program. The Turbo Pascal editor gives, as you move through a program, the current line number on the bottom left of the screen; these line numbers are in agreement with the line numbers given by the compiler on error messages.

   All programming units must be compiled in the order of their dependency. The final programming unit to be compiled should be the main program which utilizes other programming units to perform a task.

6. If the main program required no corrections and recompilations, the next step is to invoke the Ada linker which produces an executable program by putting together the separate components of the program. To link the compiled program, type the following at the prompt:

```
A:\>bamp try
```

The name of the main program (in this case "try") is the only information needed by the linker. An executable program file (given the name try.exe) is produced by the linker.

7. To run the executable program, type the name of the file (without the extension) at the prompt:

```
A:\>try
```

8. Also provided on the computers in room 104 is an interactive Ada tutor. To access this software package, type the following sequence of commands starting at the DOS C drive:

```
C:\>cd ada
C:\ada>cd adatutor
C:\ada\adatutor>ada-tutr
```

LAB NUMBER: 023

TOPIC(S) FOR LAB:
Peer reviews - extended project through preliminary design
Preliminary design review presentation

INSTRUCTIONAL OBJECTIVE(S):
1. Prc✓ide opportunity for mid-point peer review of extended project teams.
2. Students present preliminary design review for extended project.

ASSOCIATED LECTURE NUMBER:
Lecture 028

SET UP, WARM-UP:
This is the mid-point of the extended project. We want to consider your opinions in assessing the project to this point. We have prepared a peer/self evaluation instrument and will be administering it today, prior to your preliminary design review presentation.

PROCEDURE:

1. HANDOUT - Peer/self evaluation forms
   Ask each student to complete a peer/self evaluation for the overall project and one for each team on which he/she was a member.

   Stress that their responses are confidential and will be seen by the instructor only. While feedback will be provided to individuals, the confidentiality of responses will remain strictly confidential.

2. Introduce the preliminary design team and begin the review.

ASSOCIATED HANDOUTS:
Oral presentation evaluation form (2)
Material to be reviewed has been provided to reviewers in advance.

ASSOCIATED HANDOUTS:
Mid-point peer/self evaluation forms - overall project
Mid-point peer/self evaluation forms - individual teams
Oral presentation evaluation form
Material to be reviewed has been provided to reviewers in advance.

THIRD EYE PROJECT MID-POINT EVALUATION

Responses are confidential and will be seen only by the instructors. Be completely honest in rating the following from your perspective. In the scale used, UNS represents unsatisfactory and EXC represents excellent. Use back for additional comments.

Configuration management plan

```
          BELOW               ABOVE
UNS        AVG       AVG       AVG       EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Configuration manager

```
  BELOW          ABOVE
UNS       AVG       AVG       AVG       EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Requirements

```
          BELOW               ABOVE
UNS        AVG       AVG       AVG       EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Users manual

```
  BELOW          ABOVE
UNS       AVG       AVG       AVG       EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Test Plan

```
          BELOW               ABOVE
UNS        AVG       AVG       AVG       EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Preliminary design

```
  BELOW          ABOVE
UNS       AVG       AVG       AVG       EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Overall, the Third Eye team

```
          BELOW               ABOVE
UNS        AVG       AVG       AVG       EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

THIRD EYE PROJECT MID-POINT EVALUATION

TEAM: ____________________ **MEMBER:** ____________________

Complete the following for each member of the team, including yourself.

In COLUMN A, describe his/her contributions to the project.

In COLUMN B, select **VS** (very satisfied), **S** (satisfied), **D** (dissatisfied), or **VD** (very dissatisfied) to fill in the blank in the statement below.

"I am ________________ with his/her work on the team."

TEAM MEMBER	(A) CONTRIBUTION(S)	(B) SATISFACTION

LAB NUMBER: 024

TOPIC(S) FOR LAB:
User interface presentation/review
Test plan presentation/review

INSTRUCTIONAL OBJECTIVE(S):
1. User interface team presents users manual for final review.
2. Test plan team presents test plans for final review.

ASSOCIATED LECTURE NUMBER:
Lecture 029

SET UP, WARM-UP:
Remind students that the purpose of these reviews are to improve the item under review. All participants (developers, customers, and other reviewers) have a common goal: to identify issues that need to be addressed.

Remind them as well of some of the guidelines for reviews that have been discussed in the past. Include: during the review we want to identify problems, not attempt to solve them; avoid the tendency to resist change or be defensive; remember this is a team effort.

Remind them to note any issues identified that require attention and to follow up on the issues list as soon as possible.

PROCEDURE

1. Introduce the user interface team and begin the review.

2. Introduce the test plan team and begin the review.

ASSOCIATED HANDOUTS:
Oral presentation evaluation form (2)
Material to be reviewed has been provided to reviewers in advance.

LAB NUMBER: 025

TOPIC(S) FOR LAB:
Resolution of outstanding issues from last semester

INSTRUCTIONAL OBJECTIVE(S):
1. Provide issues list from last semester assessments of extended project.

ASSOCIATED LECTURE NUMBER:
Lecture 031

SET UP, WARM-UP:
As a part of the final exam last semester, students were asked individually to carefully review all of the project deliverables and identify problems or note questions on issues that they did not understand. A composite issues list has been compiled based on the student reviews as well as a thorough review by the instructors. Teams will need to resolve all of these issues before we can move on to detailed design and implementation.

PROCEDURE:

1. HANDOUT - Composite issues list
   Discuss the items and emphasize that they need to be addressed and all necessary modifications made. Students will remain on their teams from last semester. Any new students will be assigned to work with one of the teams from last semester.

2. Give teams remainder of time to work on the project.

ASSOCIATED HANDOUTS:
Composite issues list

LAB NUMBER: 026

TOPIC(S) FOR LAB:
Reorganization of extended project

INSTRUCTIONAL OBJECTIVE(S):
1. Reorganization of extended project.

ASSOCIATED LECTURE NUMBER:
Lecture 032

SET UP, WARM-UP:
Today we are going to reorganize the project to proceed with detailed design.

PROCEDURE:

1. Have configuration management team report on the state of the project artifacts, including the modifications begun in the preceding lab.

2. HANDOUT - List of project teams
   Distribute and discuss the role and responsibilities of each team. Explain that during this semester the project will be taken through implementation. The teams that will be active during this semester are configuration management, detailed design, code and unit test, and testing (and tools if a tools team is included in the project organization).

3. NOTE: Additional discussion on the project organization is included in the Projects section of this packet, in the paper on the inverted functional matrix organization.

   Discuss the immediate responsibilities of the project teams and give them the remainder of the class to work on the project.

ASSOCIATED HANDOUTS:
List of teams

EXTENDED PROJECT TEAMS

Configuration management

Detailed design

Code and unit test

Testing

Tools (optional)

LAB NUMBER: 027

TOPIC(S) FOR LAB:

Nassi-Shneiderman charts
Preparation for detailed design review

INSTRUCTIONAL OBJECTIVE(S):

1. Construct Nassi-Shneiderman charts
2. Prepare detailed design team for upcoming review

ASSOCIATED LECTURE NUMBER:

Lecture 033

SET UP, WARM-UP:

During detailed design algorithms must be designed and represented in order to be reviewed and then to be coded. We have chosen Nassi-Shneiderman charts to represent the algorithms of detailed design in the extended project. The notation was introduced in the previous lecture. Today we are going to give you practice in developing Nassi-Shneiderman charts. Then we will talk about the upcoming detailed design review.

PROCEDURE:

1. Walkthrough an example of a Nassi-Shneiderman chart. Select an algorithm with which students are familiar; e.g., select a sequential or binary search, a transaction handler for a checkbook (to handle deposits, withdrawals, or inquiries), or Mynatt exercise 2, page 236.

2. a. Separate students into their current teams and ask each to develop a Nassi-Shneiderman chart for an exchange sort algorithm (or some other with which they are familiar). Give the teams 10-15 minutes to construct a solution, remaining available for questions as they work.

   b. Review the solutions with the whole class.

3. Since the detailed design review is the first major review of the second semester, the instructor should take a few minutes to discuss reviews. Recall that the predominant method of design evaluation is the design review. At a design review, either preliminary design or detailed design, there are two basic questions to keep in mind.

   Does the design fulfill the requirements?
   Does the design meet established design standards?

   Review the material first presented in Lab 006. This is particularly important if there are any students in the class who were not in the first semester.

4. HANDOUT - Detailed design review form
   Distribute and discuss the detailed design review form. This will be used by reviewers to provided feedback to the presenters.

   Remind the detailed design team that the material to be reviewed must be provided to reviewers in advance. Make arrangements for the materials to be provided to the instructors for duplication and then made available to the reviewers.

<u>ASSOCIATED HANDOUTS</u>:

Detailed design review form
Suggestions for giving and oral presentation

Detailed Design Review Form

Project Name :________________________

Reviewer Name :________________________

I. High Level Issues

A: Requirements: any requirements missed, requirements over-worked?

B: Design: suggestions for improvement of architecture or procedures; other strategies

C: The Design fits the whole specification including quality standards such as flexibility, friendliness, efficiency, and cost effective.

II Design Deliverable Details

A: Revised Test Plan: items over tested or under-tested, suggested tests

B: Design Model: good use of notation, clear model, suggested improvements

III Detailed Design

A: Can design be implemented easily: availability of adequate programming and testing manpower. Adequate hardware facilities-computer, data storage...

B: Is the design programmable- does not require exotic functions

C: Is there a suggested or obvious order of implementation or approximate times for the development of and description of the production relations between the modules. What is the order of need for equipment required to implement the design.

D: Comments on other deliverables

LAB NUMBER: 028

TOPIC(S) FOR LAB:
Detailed design review presentation

INSTRUCTIONAL OBJECTIVE(S):
1. Students present detailed design review for extended project.

ASSOCIATED LECTURE NUMBER:
Lecture 038

SET UP, WARM-UP:

As was discussed in the last lab, today's detailed design review is very important. You want to assure that the design fulfills the requirements, meets standards, and is consistent with the preliminary design. This assurance is critical since the preliminary design is the base on which your implementation will rest. Problems uncovered will be much easier to resolve now than they will be during implementation or system testing.

PROCEDURE:

1. Remind the detailed design team that they should note any issues which arise during the review. Each item on this "issues list" must be addressed and appropriate modifications made where needed. The issues list thus serves as action item checklist for the team as they address the issues.

   The instructor should maintain his/her own issues list as a means of establishing a follow-up procedure to assure that the items are addressed.

   Instructors should maintain their role as customer as much as possible, reverting to role of instructor only when necessary for such things as maintaining the schedule, reminding participants of the purpose and/or ground rules, and maintaining order. Critiques should be saved until the next lecture or lab.

2. Introduce the detailed design team and begin the review.

ASSOCIATED HANDOUTS:
Oral presentation evaluation form
Material to be reviewed has been provided to reviewers in advance.

LAB NUMBER: 029

TOPIC(S) FOR LAB:
Feedback on detailed design review presentation.

INSTRUCTIONAL OBJECTIVE(S):
1. Provide feedback on detailed design review presentations.

ASSOCIATED LECTURE NUMBER:
Lecture 040

SET UP, WARM-UP:

At the last lab the detailed design was reviewed. We want to provide you with our reactions to the review and make sure all issues have been identified and are being addressed.

PROCEDURE:

1. Provide specific feedback on the detailed design. Specifically ask about the "issues list" which the detailed design team should have compiled during the review. (Use your own issues list as a check.) Ask how each item was addressed and the disposition of each.

   This meeting is critical to baseline the detailed design. Teams are about to begin implementation and agreement must be reached on what is to be implemented.

ASSOCIATED HANDOUTS:

LAB NUMBER: 030

TOPIC(S) FOR LAB:
Code inspections

INSTRUCTIONAL OBJECTIVE(S):
1. Understand haw to conduct a code inspection.

ASSOCIATED LECTURE NUMBER:
Lecture 041

SET UP, WARM-UP:

Reviews have been used throughout the software development phases up to this point as a software quality assurance activity. Reviews will continue to be used in the implementation phase. Today we want to consider code reviews, or code inspections.

PROCEDURE:

1. Introduce the video-tape provided in the educational materials package from the Software Engineering Institute [L.E. Deimel, "Scenes from Software Inspections," CMU/SEI-91-5]. The package includes a video-tape "Scenes of Software Inspections" and discussion aids. In less than 20 minutes, students see several dramatizations of common pitfalls in formal reviews. The presentation makes the pitfalls and the problems they generate obvious to the students. Each dramatization is intended to be followed by a discussion of how to avoid these pitfalls. This discussion reduces anxiety about reviews and develops an appreciation of appropriate review roles and behavior.

2. The code and unit test team will be required to conduct code inspections of their work at the appropriate time.

ASSOCIATED HANDOUTS:

LAB NUMBER: 031

TOPIC(S) FOR LAB:

Introduction to maintenance project
Team organization for maintenance project
Maintenance assignment 1

INSTRUCTIONAL OBJECTIVE(S):

1. Introduce maintenance project and project organization.
2. Assign maintenance exercise 2.

ASSOCIATED LECTURE NUMBER:

Lecture 045

SET UP, WARM-UP:

In practice software engineers often work on multiple projects simultaneously. It is not unusual that while a development project is underway, maintenance must be performed on an existing system. Today we are going to begin on a maintenance project which will overlap the extended project.

PROCEDURE

1. Discuss the organization of the maintenance project.

   a. For the duration of the maintenance project all class meetings, including both lecture and lab time, will be devoted to an in-class maintenance project. We want to prevent you from working on this maintenance project outside of class. Of course you will continue working on the extended project outside of class time.

   b. HANDOUT - Maintenance project teams
   Discuss the chief programmer organization. The project organization for the maintenance project will be chief-programmer teams. We have identified the chief programmers.

2. Handout - The DASC Software System
Introduce the DASC system. Give an overview of what it does and the documents that are available.

3. Explain that the DASC system works, but not in our environment. The maintenance assignments will involve:

   a. porting it to our environment;

   b. testing it according to the DASC test plan and test data provided, and completing Discrepancy Reports as necessary;

   c. performing some maintenance (enhancement) tasks based on

Change Requests to be provided.

4. HANDOUT - DASC Maintenance exercise 1
Distribute and discuss maintenance exercise 1. Also make available to each team DASC documentation including the users manual, requirements document, test cases, expected results for test cases, and discrepancy report forms.

ASSOCIATED HANDOUTS:
Rosters for maintenance project teams (with Chief Programmers identified)
DASC Maintenance exercise 1 and associated materials

DASC Maintenance Exercise 1

The DASC system has recently been installed in our lab. The purpose of this maintenance exercise is to run DASC on a test suite and record any discrepancies for consideration by a Change Control Board. The test suite and discrepancy report forms are provided. Specific directions for completing the exercise follow.

1. Use the disks provided. Each contains:

   a. DASC executable, STYLE_CH, in root directory.

   b. test suite, in subdirectory TEST. (The test suite consists of a collection of Ada programs to be used as input to DASC.)

   c. COMMANDL.TXT file, in root directory.

   d. DASC source code, in root directory. (The DASC source code is <u>not needed</u> for this exercise.)

2. Execute STYLE_CH on <u>each</u> program in the test suite:

   a. Prior to executing STYLE_CH, edit COMMANDL.TXT so that it contains the complete path name of the test program.

   b. run STYLE_CH. WARNING: DASC always gives an "unable to open" error message. Ignore it.

3. For each test, STYLE_CH creates two output report files, a flaw report (testprogname.FLW) and a style report (testprogname.STY). Compare the results of these reports with the expected results distributed and file a discrepancy report for each discrepancy found.

<u>LAB NUMBER</u>: 032

<u>TOPIC(S) FOR LAB</u>:
Code inspections

<u>INSTRUCTIONAL OBJECTIVE(S)</u>:
1. Code and unit test team conducts code inspections.

<u>ASSOCIATED LECTURE NUMBER</u>:
Lecture 046

<u>SET UP, WARM-UP</u>:

In todays lab the code and unit test team is going to conduct code inspections of their work. We will simply be observers during this.

<u>PROCEDURE</u>:

1. The code and unit test team conducts code inspections. The other teams are given this time to work on their parts of the project. The instructors function primarily as observers and intervene in the code inspections only if necessary.

<u>ASSOCIATED HANDOUTS</u>:

LAB NUMBER: 033

TOPIC(S) FOR LAB:
Feedback on Maintenance exercise 1
Maintenance exercise 2

INSTRUCTIONAL OBJECTIVE(S):
1. Provide feedback on maintenance exercise 1.
2. Assign maintenance exercise 2.

SET UP, WARM-UP:
The first maintenance exercise should have given you an external (user) view of the DASC system; specifically, the system's output (flaw and style reports), the user interface, and the types of "style factors" the system is assessing. It should serve as a nice lead-in to the next DASC maintenance exercise that you will be given today.

PROCEDURE

1. Discuss results of maintenance exercise 1.

2. HANDOUT - DASC maintenance exercise 2
   HANDOUT - DASC source code

   Discuss the exercise. We are providing you with some internal documentation of the system and asking you, as maintenance teams, to plan needed changes due to the attached discrepancy reports and change requests.

ASSOCIATED HANDOUTS:
DASC Maintenance exercise 2 and attachments

DASC Maintenance Exercise 2

Attached are two DASC Discrepancy Reports and one DASC Change Request. Each of the three has been approved by the Configuration Control Board.

Provide a detailed description of the modifications necessary to correct the problems. Specifically, submit:

a) changes to the design documents provided with this exercise; and

b) changes to the source code provided with this exercise.

At this point you are not required to submit plans for testing the system after the modifications are made.

Attachments:	DASC Discrepancy Report 37
	DASC Discrepancy Report 42
	DASC Change Request 11

DASC TEST DISCREPANCY REPORT **Report No.: 37**

Originator (Team): **3** Date: **8/7/93**

Test Case/Program: **All**

Description of Expected Result:

Description of Actual Result:

During execution, the message "file cannot be opened" is always displayed to the screen.

Additional Comments:

The extraneous message has no apparent significance, and presently users have been instructed to ignore it.

RESOLUTION (to be completed by CCB)

___ Change Required

___ Waived - Describe reasons waived:

[✓ X] Approved For Analysis

___ Duplicate Problem - Associated Test Discrepancy Report No(s):

CM Signature: Date:

DASC TEST DISCREPANCY REPORT **Report No.: 42**

Originator (Team): **3** Date: **8/7/93**

Test Case/Program: **Test 223**

Description of Expected Result:

If the COMMANDL.TXT input file is empty or contains the name of a non-existent Ada source program, DASC should detect this and respond to the user in some appropriate manner.

Description of Actual Result:

If the COMMANDL.TXT input file is empty or contains the name of a non-existent Ada source program, several exceptions are raised and the DASC system fails to perform properly. See attached print-screen of DASC display whenever this situation occurs.

Additional Comments:

RESOLUTION (to be completed by CCB)

___ Change Required

___ Waived - Describe reasons waived:

X Approved For Analysis

___ Duplicate Problem - Associated Test Discrepancy Report No(s):

CM Signature: Date:

DASC CHANGE REQUEST Change Request No.: 11

Originator: **Bob Dorsey, User Support (Dept 3287)** Date: **8/8/93**

Change Type: X New Feature __ Cost Reduction __ Other (describe)

Change description:

Currently DASC expects the name of the Ada source program that is to be processed to be in the file COMMANDL.TXT. Add an option that allows the user to enter, directly from the keyboard at execution time, the filename of the Ada source program that is to be processed.

The system should continue to support the use of the COMMANDL.TXT file.

CCB Decision (to be completed by CCB)

X Approved As Is

___ Approved With Modification

___ Waived

Describe reasons waived or modification:

CM Signature: Date:

LAB NUMBER: 034

TOPIC(S) FOR LAB:
Feedback on Maintenance exercise 2
Maintenance exercise 3

INSTRUCTIONAL OBJECTIVE(S):
1. Provide feedback on maintenance exercise 2.
2. Assign maintenance exercise 3.

SET UP. WARM-UP:
In the last maintenance exercise you responded to discrepancy reports and a change request for DASC. Today we want you to respond to plan some modifications to enhance the system. Again, these will be provided in the form of change requests.

PROCEDURE

1. Discuss results of maintenance exercise 2.

2. HANDOUT - DASC maintenance exercise 3

   Discuss the exercise.

ASSOCIATED HANDOUTS:
DASC Maintenance exercise 3 and attachments

DASC Maintenance Exercise 3

Plan the modifications necessary to enhance the system as called for in DASC Change Requests 23 and 39.

Submit the following for each change request.

a) Give a narrative high-level description of two distinct solutions to the problem.

b) For each of the solutions, give arguments for and against.

c) For the solution you choose as best, show:

   (1) the changes to the design documents provided; and

   (2) the screen layouts for any user interface (Change Request 23), and input file specifications (Change Request 39); and

   (3) indicate what sections of the source code have to be changed and how.

Att: DASC Change Request 23
DASC Change Request 39

DASC CHANGE REQUEST **Change Request No.: 23**

Originator: **Bob Dorsey, User Support (Dept 3287)** Date: **8/10/93**

Change Type: [✓ New Feature] __ Cost Reduction __ Other (describe)

Change description:

Modify DASC to allow screen display of flaw and style reports. The system should ask the user if he/she wants to see a display of the flaw report. If so, the flaw report should be displayed on the screen one "screen-full" at a time (like the DOS *more* command). After each screen-full, the user should be able to request the next screen-full or exit. Similarly, a display of the style report should be allowed.

CCB Decision (to be completed by CCB)

[✓ Approved As Is]

___ Approved With Modification

___ Waived

Describe reasons waived or modification:

CM Signature: Date:

DASC CHANGE REQUEST **Change Request No.: 39**

Originator: **Jodie Milosovich, User Support (Dept 3287)** Date: **8/10/93**

Change Type: X New Feature __ Cost Reduction __ Other (describe)

Change description:

Modify DASC so that the threshold values of the quantifiable style parameters are read from am input file rather than being hard-coded into the system. This will allow different organizations to customize the system more easily.

CCB Decision (to be completed by CCB)

X Approved As Is

___ Approved With Modification

___ Waived

Describe reasons waived or modification:

CM Signature: Date:

LAB NUMBER: 035

TOPIC(S) FOR LAB:
Final peer/self assessment - extended project
System acceptance test/review - extended project

INSTRUCTIONAL OBJECTIVE(S):
1. Administer final peer/self assessment of extended project teams.
2. Conduct system acceptance test/review for extended small project.

ASSOCIATED LECTURE NUMBER:

SET UP, WARM-UP:
As we have been doing throughout the team projects, we want to consider your opinions in assessing the extended project teams. All of the project deliverables are due today and all that remains is the acceptance test and addressing any issues that arise out of this review. Today you will be completing a peer/self assessment instrument for the extended project and for the teams you have been working on in the latter stages of the project. Immediately following that we will conduct the acceptance test presentations.

PROCEDURE:
1. HANDOUT - Final peer/self evaluation forms for extended project.

   Ask each student to complete a peer/self evaluation form for the teams on which he/she participated and for the overall project. Stress that their responses are confidential and will be seen by the instructor only. While composite feedback will be provided to individuals, the confidentiality of responses will be strictly maintained.

ASSOCIATED HANDOUTS:
Peer/self evaluation forms for extended project
Oral presentation evaluation form
Material to be reviewed has been provided to reviewers in advance.

THIRD EYE - END OF PROJECT PEER EVALUATION

Responses are confidential and will be seen only by the instructors.

1. Rate the following. UNS represents unsatisfactory and EXC represents excellent.

Configuration management plan

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Requirements

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Test Plan

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Users manual

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Preliminary Design

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Detailed design

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Code and unit test

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Testing

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Configuration Management

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

Overall, the Third Eye System

```
        BELOW              ABOVE
UNS      AVG      AVG       AVG      EXC
|-----|-----|-----|-----|-----|-----|-----|-----|
```

COMMENTS:

2. Characterize the interactions between the indicated teams using the following scale by circling the most the most appropriate descriptor.

VN = Very Non-productive A = Adequate VP = Very Productive
N = Non-productive P = Productive

a) Preliminary design team & Detailed design team ----- VN N A P VP

b) Detailed design team & Code and unit test team ----- VN N A P VP

c) Detailed design team & Testing team ------------------- VN N A P VP

d) Code & unit test team & Testing team ----------------- VN N A P VP

e) Detailed design team & Configuration manager ------- VN N A P VP

f) Code and unit test team & Configuration manager ---- VN N A P VP

g) Testing team & Configuration manager ---------------- VN N A P VP

THIRD EYE - END OF PROJECT PEER EVALUATION

TEAM: ________________ MEMBER: ________________

Complete the following for each member of the team, including yourself.

In COLUMN A, describe his/her contributions to the project.

In COLUMN B, select **VS** (very satisfied), **S** (satisfied), **D** (dissatisfied), or **VD** (very dissatisfied) to fill in the blank in the statement below.

"I am ______________ with his/her work on the team."

TEAM MEMBER	(A) CONTRIBUTION(S)	(B) SATISFACTION

4. Imagine that $2000 in bonuses is to be distributed among the THIRD EYE Project team members. Half of it ($1000) is to be distributed based on the intellectual contribution to the project, i.e., significant ideas and solutions contributed. The other half ($1000) is to be distributed based on amount of individual effort contributed to the project.

   Distribute the bonuses. If you wish, justify each of the assignments. Be very specific; list some especially significant contributions for which the team member should be proud or where the project was made more or less difficult because of it.

Project Member Name	$1000 concepts	$1000 effort	JUSTIFICATION

LAB NUMBER: 036

TOPIC(S) FOR LAB:
Instructors' assessment of extended project.

INSTRUCTIONAL OBJECTIVE(S):
1. Provide assessment of extended project.
2. Provide assessment of individual team members.

ASSOCIATED LECTURE NUMBER:

SET UP, WARM-UP:
Recall that the course policies explained that there were three factors to be considered in your individual project grades: the product produced by the team, peer assessment of individual contributions to the team effort, and the instructor(s)' assessment of individual contributions. Today each of you will receive a copy of our assessment of your project with a team project grade and with an individual grade for your contribution to the project.

PROCEDURE:
1. HANDOUT - Extended project evaluation
   Distribute individual assessment reports containing the product grade, the individual grade, and extended comments on the product.

2. Invite teams or individuals to make appointments to discuss the assessments. Stress that we will discuss the peer/self assessments with individuals but only in composite terms; confidentiality will be maintained.

ASSOCIATED HANDOUTS:
Extended project evaluations

EXTENDED PROJECT EVALUATION: THIRD EYE PROJECT

TEAM MEMBER: ____________________

TEAM PRODUCT GRADE: 95 **TEAM MEMBER'S GRADE:** ____

COMMENTS ON DELIVERED PRODUCT

Most of the following comments pertain to the delivered and demonstrated software product and are not necessarily reflective of the time and/or effort expended. There is no question the time and effort expended by the project team was superlative. Overall, we are extremely pleased with the amount and quality of the work of the Third Eye project team. The comments below are suggestions for improvement and are not intended to detract from our overall satisfaction with your work.

OVERALL

Problems observed with the delivered product include the following.

1. User interface - The user interface appeared to have "fallen between the cracks" throughout the entire project and, subsequently, was very crude in the final product.

2. Inconsistencies exist both between and among the requirements, preliminary design, detailed design, and code documents. These fall into two categories: first, decisions documented by one team not being followed by another, and second, changes made by one team not being reflected (updated) in earlier documents.

3. Problems with filters 3 and 4 were evident in the acceptance test. In filter 3, the user selection of specific procedures/functions for further evaluation was cumbersome. Similarly the results of filter 4 needed improvement (e.g. results unclear, "phantom" functions/procedures, ...).

CONFIGURATION MANAGEMENT

The configuration management plan was feasible and complete. Its implementation was visible and worked well under the circumstances. Good control mechanisms ("manilla envelope system" in lab, and memos summarizing approved changes and new baselines) were developed and well-managed, but unfortunately are not documented in the configuration management plan. The final build (packaging) of the Third Eye project was excellent. One suggestion for improvement would be a better method for the reader to access documents in the configuration management plan (perhaps global page numbering or labeling and numbering within appendices). The modification history was inconsistent in documents (e.g. chronologically forward in some documents, backward in others).

REQUIREMENTS

The requirements for this project were difficult to pin down and overall a pretty good job was done and the requirements were complete and appropriate. As the project progressed and requirements changed, the changes were not always made in the requirements documents. For example the data dictionary was not updated; nor was the narrative description (also has a couple of typos). Important changes to the non-functional requirements were made in the requirements list but the modifications are not clearly traceable through the modification history (e.g. references to RQ1, RQ 2, etc). The process specifications of the structured analysis model were helpful to subsequent life-cycle teams.

TEST PLAN

The test plan was complete and well organized, and appropriate detail was provided. It provided a solid framework for the testing team to later use. It is cumbersome to integrate all of the filled-out forms (hard to trace what is done).

USER MANUAL

The user manual format and first draft were excellent, but the final manual does not reflect the current system.

PRELIMINARY DESIGN

Overall a good job was done though there are some inconsistencies in the preliminary design as well as between preliminary design and detailed design. The object diagram of the object model shows inheritance from the class Filter to the subclasses Filter 1, ..., Filter 8, yet it doesn't exist (the relationship should be shown as an aggregation); attributes were removed from the filters and we asked that they continue to be shown; without attributes there is no information to be found when the filters are called by the summary report. The Ada specifications are inconsistent with the object dictionary (e.g. Summary Report depicted as reading filters but the filters don't hold the values; also confusion with Print Report File).

DETAILED DESIGN

As with preliminary design, overall a good job was done though there are some inconsistencies in the detailed design as well as between detailed design and preliminary design (e.g. Summary report output file) and between detailed design and coding. The output file for summary report file structure is missing; Filter PD-4 will not work as specified; there needs to be two subprograms named to be compared.

CODING

Overall a good job was done, particularly considering the time constraints and resource problems. There are inconsistencies between detailed design and the source code (e.g. summary report). The Ada standards were not adhered to consistently.

LAB NUMBER: 037

TOPIC(S) FOR LAB:
Function points

INSTRUCTIONAL OBJECTIVE(S):

1. Be able to estimate resources for a project based on function points analysis.

ASSOCIATED LECTURE NUMBER:
Lecture 049

SET UP, WARM-UP:
During the earlier lecture we discussed COCOMO and function points analysis as methods to estimate resources (time to develop, personnel needed at various phases) needed for a project. The COCOMO model relies on a single independent variable, estimated lines of code, to derive its estimates. Function points analysis provides a means of estimating the lines of code based on the requirements of the system and the complexity of its development. Today we're going to have you estimate lines of code for the extended project using function points analysis.

PROCEDURE:

1. Begin by considering only the user inputs for the extended project. Lead the class through the completion of function points worksheets. Identify the number of user inputs and then discuss modifying these using the technical complexity factors.

2. HANDOUT - Function points work sheets (see lecture 049).
   Divide the class into groups of 3-5 students each. Ask each group to repeat the exercise above with each group assigned one of the following:
   a. user outputs;
   b. user inquiries;
   c. files;
   d. external interfaces.

   NOTE: This is probably best done as a group exercise outside of class.

3. Collect the group results and compute the total number of function points for the extended project. Us this to derive the estimated number of lines of source code.

4. Consider how this estimate compares with the actual lines of code for the extended project. Discuss reasons for the discrepancy between the estimated and actual lines of code.

ASSOCIATED HANDOUTS:
Function points work sheets (see lecture 049).

LAB NUMBER: 038

TOPIC(S) FOR LAB:
Ethical issues and professionalism

INSTRUCTIONAL OBJECTIVE(S):

1. Be able to identify ethical issues in software engineering
2. Understand professional responsibilities of software developers
3. Be able to analyze ethical scenarios to identify the ethical issues, discover applicable ethical principles, and make ethical decisions.

ASSOCIATED LECTURE NUMBER:
Lecture 050

SET UP, WARM-UP:

NOTES TO INSTRUCTOR:

1. Several laboratories concerning ethical issues are described. These labs, as well as lectures on ethical issues and professionalism, are appropriate at various times during the two-semester course. Placing them in the last several weeks of the second semester takes advantage of the accumulated student project experience and serves as a capstone. Students are less likely to be preoccupied with project work at this point and may be more attentive to the topic. Introducing this material immediately after completion of the small projects or at the end of preliminary design of the extended project (either at the end of first semester or the beginning of the second semester) also has merit. At these points students have had meaningful project experience but plenty of project work remains. Introducing the topic here with followup throughout the second semester will cause students to think about ethical issues and professionalism during their subsequent project work. Yet another option is to introduce this material during the second semester coincident with the extended project reorganization as detailed design completes and implementation activities begin.

2. Ethical Decision Making and Information Technology by Kallman and Grillo (McGraw-Hill, 1993) and the accompanying instructor's manual are excellent resources for analyzing ethical issues. They present a four-step analysis process and a worksheet for making appropriate decisions. We highly recommend this as a supplementary textbook. If adopted then the worksheets and other supporting materials may be reproduced for use with the textbook. Several of the laboratory exercises described below assume that the Kallman/Grillo textbook has been adopted.

PROCEDURE and ASSOCIATED HANDOUTS:
Attached are Procedure and Associated Handouts sections for a number of suggested laboratories. These are referred to as 038-1, 038-2, etc.

PROCEDURE: 038-1

NOTE: This can be done as an individual exercise or a group exercise involving the current project teams.

It is assumed that the associated handouts listed below have already been assigned as outside reading.

1. Describe the following. Some software development companies have been known to develop a first version of a software package and begin selling it even though they are aware that it still has some "bugs". They expect that users will find and report bugs as well as make other suggestions for improving the product. The software development company plans to use these bug reports (complaints) and other suggestions to improve the product for "version 1.1".

2. Ask whether any of the Cases in the article Using the New ACM Code of Ethics in Decision Making address this issue? Expect that students will recognize the similarities with Case 6.

3. Ask students to analyze the scenario (as in done in the article) and cite specific imperatives in the ACM Code of Ethics and Professional Conduct that are applicable.

4. If this was done by teams, have each team report on its analysis.

ASSOCIATED HANDOUTS:

ACM Code of Ethics and Professional Conduct

Article: "Using the New Acm Code of Ethics in Decision Making", by Anderson, Johnson, Gotterbarn, and Perrolle. Communications of the ACM, Feb 1993

PROCEDURE: 038-2

NOTE: This can be done as an individual exercise or a group exercise involving the current project teams.

It is assumed that the associated handouts listed below have already been assigned as outside reading.

1. Describe the following scenario adapted from Ethical Conflicts in Information and Computer Science, Technology, and Business by Parker, D., Swope, S., and Baker, B. (Wellesley, MA: QED Information Sciences, 1990).

   The ETSU Software Corporation is developing the hardware and software for a computerized voting machine under a contract with the Freedom Tabulating Company (FTC), which is marketing the system. FTC has persuaded several cities and states to purchase the system for use in the next elections. Sandy Smith, a software engineer with ETSU, is aware of problems in the hardware/software interface that are likely to cause the machine to miscount approximately one-half of one percent (0.5%) of the time. Sandy reports this back to the software project director, who responds "that's a hardware problem".

2. Ask whether any of the Cases in the article Using the New ACM Code of Ethics in Decision Making address this issue?

3. a. Ask students to analyze the scenario (as is in done in the article) and cite specific imperatives in the ACM Code of Ethics and Professional Conduct that are applicable.
   b. What responsibility, if any, does Sandy have for the ultimate use of the system? Cite the specific aspects of the code that apply.

4. If this was done by teams, have each team report on its analysis.

ASSOCIATED HANDOUTS:

ACM Code of Ethics and Professional Conduct

Article: "Using the New Acm Code of Ethics in Decision Making", by Anderson, Johnson, Gotterbarn, and Perrolle. Communications of the ACM, Feb 1993

PROCEDURE: 038-3

NOTE: This can be done as an individual exercise or a group exercise involving the current project teams.

It is assumed that the associated handouts listed below have already been assigned as outside reading.

1. Describe the following. K. Kuhn is a software engineer for SECURE Software. SECURE has developed a software system that has been successfully tested and is now being used in several states. The system enables the criminal justice system to monitor low-risk prisoners and has helped solve the "overcrowded prison" problem (by releasing low-risk offenders but monitoring their movements and the people they associate with, and by enforcing curfews). SECURE has now been approached by the government of a foreign country and asked to design a system based on that used in the United States but with a number of changes. K. Kuhn has raised concerns with SECURE that the foreign government intends to use the requested system to more effectively and efficiently manage and preserve their oppressive practices towards their citizens. Kuhn's management takes the position that it is none of their business how the foreign government uses the system.

2. Ask whether any of the Cases in the article Using the New ACM Code of Ethics in Decision Making address this issue?

3. a. Ask students to analyze the scenario (as in done in the article) and cite specific imperatives in the ACM Code of Ethics and Professional Conduct that are applicable.
   b. Kuhn is considering many actions: doing nothing; refusing to work on the project; resigning; expressing the concerns to higher management; informing the press of the foreign government's plans for the system and SECURE's role in it; and contacting the Congressional representative for the district. What guidance, if any, does the proposed ACM Code of Ethics and Professional Conduct provide Kuhn? In your answer, cite the specific sections of the code that apply.
   c. What responsibility, if any, does Kuhn have for the ultimate use of the system? Cite the specific aspects of the code that apply.

4. If this was done by teams, have each team report on its analysis.

ASSOCIATED HANDOUTS:
ACM Code of Ethics and Professional Conduct
Article: "Using the New Acm Code of Ethics in Decision Making", by Anderson, Johnson, Gotterbarn, and Perrolle. Communications of the ACM, Feb 1993

PROCEDURE: 038-4

NOTE: This can be done as an individual exercise or a group exercise involving the current project teams.

It is assumed that the associated handouts listed below have already been assigned as outside reading.

1. Describe the following. According to the Compliance with Code section, each ACM member agrees to personally uphold the principles of the code. In your opinion, does the code place any responsibility on members regarding violations of the code by other members? If so, what is the responsibility? If not, do you think this is a serious omission in the code?

2. If this was done by teams, have each team report on its analysis.

ASSOCIATED HANDOUTS:

ACM Code of Ethics and Professional Conduct

Article: "Using the New Acm Code of Ethics in Decision Making", by Anderson, Johnson, Gotterbarn, and Perrolle. Communications of the ACM, Feb 1993

PROCEDURE: 038-5

ASSOCIATED HANDOUTS:

ACM Code of Ethics and Professional Conduct
Article: Using the New Acm Code of Ethics in Decision Making by Anderson, Johnson, Gotterbarn, and Perrolle. Communications of the ACM, Feb 1993
Kallman's Four Step Analysis Process
Case Worksheets for analysis of ethical scenarios (Kallman)

1. HANDOUT - Kallman's Four Step Analysis Process
   Discuss Kallman's process to analyze an ethical situation. Lead the class through the analysis of a scenario as an example.

2. HANDOUT - Case Worksheets for analysis of ethical scenarios (Kallman)
   Describe an ethical scenario (e.g. any of those used in Procedures 038-1, 038-2, 038-3, 038-4, or one of your own.)

3. Ask students individually to follow Kallman's four-step analysis process to complete a worksheet for the scenario. While this can be done in class, we prefer to provide the worksheet and scenario to the class in advance and students are expected to come to the lab with the worksheet completed.

4. Divide students into groups consisting of the current project teams. Give the group 20-30 minutes to analyze the scenario as a group and complete a group (consensus) worksheet. (Note - Kallman's textbook, Ethical Decision Making and Information Technology and the accompanying instructor's manual has suggestions for conducting such exercises.)

5. Have each team report on its analysis.

ASSOCIATED HANDOUTS:

ACM Code of Ethics and Professional Conduct
Article: Using the New Acm Code of Ethics in Decision Making by Anderson, Johnson, Gotterbarn, and Perrolle. Communications of the ACM, Feb 1993
** Kallman/Grillo Four Step Analysis Process
** Kallman/Grillo Case Worksheets for analysis of ethical scenarios

** - see Note 2 in Lab SET UP, WARM-UP section

ACM Code of Ethics and Professional Conduct *

Preamble. Commitment to ethical professional conduct is expected of every member (voting members, associate members, and student members) of the Association for Computing Machinery (ACM).

This Code, consisting of 24 imperatives formulated as statements of personal responsibility, identifies the elements of such a commitment. It contains many, but not all, issues professionals are likely to face. Section 1 outlines fundamental ethical considerations, while Section 2 addresses additional, more specific considerations of professional conduct. Statements in Section 3 pertain more specifically to individuals who have a leadership role, whether in the workplace or in a volunteer capacity such as with organizations like ACM. Principles involving compliance with this Code are given in Section 4.

The Code shall be supplemented by a set of Guidelines, which provide explanation to assist members in dealing with the various issues contained in the Code. It is expected that the Guidelines will be changed more frequently than the Code.

The Code and its supplemented Guidelines are intended to serve as a basis for ethical decision making in the conduct of professional work. Secondarily, they may serve as a basis for judging the merit of a formal complaint pertaining to violation of professional ethical standards.

It should be noted that although computing is not mentioned in the imperatives of section 1.0, the Code is concerned with how these fundamental imperatives apply to one's conduct as a computing professional. These imperatives are expressed in a general form to emphasize that ethical principles which apply to computer ethics are derivsd from more general ethical principles.

It is understood that some words and phrases in a code of ethics are subject to varying interpretations, and that any ethical principle may conflict with other ethical principles in specific situations. Questions related to ethical conflicts can best be answered by thoughtful consideration of fundamental principles, rather than reliance on detailed regulations.

1. GENERAL MORAL IMPERATIVES. As an ACM member I will . . .

1.1 Contribute to society and human well-being.
1.2 Avoid harm to others.
1.3 Be honest and trustworthy.
1.4 Be fair and take action not to discriminate.
1.5 Honor property rights including copyrights and patents.
1.6 Give proper credit for intellectual property.
1.7 Respect the privacy of others.
1.8 Honor confidentiality.

----

* Adopted by ACM Council 10/16/92.

2. MORE SPECIFIC PROFESSIONAL RESPONSIBILITIES. As an ACM computing professional I will . . .

2.1 Strive to achieve the highest quality, effectiveness and dignity in both the process and products of professional work.
2.2 Acquire and maintain professional competence.
2.3 Know and respect existing laws pertaining to professional work.
2.4 Accept and provide appropriate professional review.
2.5 Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks.
2.6 Honor contracts, agreements, and assigned responsibilities.
2.7 Improve public understanding of computing and its consequences
2.8 Access computing and communication resources only when authorized to do so.

3. ORGANIZATIONAL LEADERSHIP IMPERATIVES. As an ACM member and an organizational leader, I will . . . . .

3.1 Articulate social responsibilities of members of an organizational unit and encourage full acceptance of those responsibilities.
3.2 Manage personnel and resources to design and build information systems that enhance the quality of working life.
3.3 Acknowledge and support proper and authorized uses of an organization's computing and communication resources.
3.4 Ensure that users and those who will be affected by a system have their needs clearly articulated during the assessment and design of requirements; later the system must be validated to meet requirements.
3.5 Articulate and support policies that protect the dignity of users and others affected by a computing system.
3.6 Create opportunities for members of the organization to learn the principles and limitations of computer systems.

4. COMPLIANCE WITH THE CODE. As an ACM member, I will . . . .

4.1 Uphold and promote the principles of this Code.
4.2 Treat violations of this code as inconsistent with membership in the ACM.

GUIDELINES

1. GENERAL MORAL IMPERATIVES. As an ACM member I will ....

1.1 Contribute to society and human well-being.

This principle concerning the quality of life of all people affirms an obligation to protect fundamental human rights and to respect the diversity of all cultures. An essential aim of computing professionals is to minimize negative consequences of computing systems, including threats to health and safety. When designing or implementing systems, computing professionals must attempt to ensure that the products of their efforts will be used in socially responsible ways, will meet social needs, and will avoid harmful effects to health and welfare.

In addition to a safe social environment, human well-being includes a safe natural environment. Therefore, computing professionals who design and develop systems must be alert to, and make others aware of, any potential damage to the local or global environment.

1.2 Avoid harm to others.

"Harm" means injury or negative consequences, such as undesirable loss of information, loss of property, property damage, or unwanted environmental impacts. This principle prohibits use of computing technology in ways that result in harm to any of the following: users, the general public, employees, employers. Harmful actions include intentional destruction or modification of files and programs leading to serious loss of resources or unnecessary expenditure of human resources such as the time and effort required to purge systems of "computer viruses."

Well-intended actions, including those that accomplish assigned duties, may lead to harm unexpectedly. In such an event the responsible person or persons are obligated to undo or mitigate the negative consequences as much as possible. One way to avoid unintentional harm is to carefully consider potential impacts on all those affected by decisions made during design and implementation.

To minimize the possibility of indirectly harming others, computing professionals must minimize malfunctions by following generally accepted standards for system design and testing. Furthermore, it is often necessary to assess the social consequences of systems to project the likelihood of any serious harm to others. If system features are misrepresented to users, coworkers, or supervisors, the individual computing professional is responsible for any resulting injury.

In the work environment the computing professional has the additional obligation to report any signs of system dangers that might result in serious personal or social damage. If one's superiors do not act to curtail or mitigate such dangers, it may be necessary to "blow the whistle" to help correct the problem or reduce the risk. However, capricious or misguided reporting of violations can, itself, be harmful. Before reporting violations, all relevant aspects of the incident must be thoroughly assessed. In particular, the assessment of risk and responsibility must be credible. It is suggested that advice be sought from other computing professionals. See principle 2.5 regarding thorough evaluations.

1.3 Be honest and trustworthy.

Honesty is an essential component of trust. Without trust an organization cannot function effectively. The honest computing professional will not make deliberately false or deceptive claims about a system or system design, but will instead provide full disclosure of all pertinent system limitations and problems.

A computer professional has a duty to be honest about his or her own qualifications, and about any circumstances that might lead to conflicts of interest.

Membership in volunteer organizations such as ACM may at times place individuals in situations where their statements or actions could be interpreted as carrying the "weight" of a larger group of professionals. An ACM member will exercise care to not misrepresent ACM or positions and policies of ACM or any ACM units.

1.4 Be fair and take action not to discriminate.

The values of equality, tolerance, respect for others, and the principles of equal justice govern this imperative. Discrimination on the basis of race, sex, religion, age, disability, national origin, or other such factors is an explicit violation of ACM policy and will not be tolerated.

Inequities between different groups of people may result from the use or misuse of information and technology. In a fair society,all individuals would have equal opportunity to participate in, or benefit from, the use of computer resources regardless of race, sex, religion, age, disability, national origin or other such similar factors. However, these ideals do not justify unauthorized use of computer resources nor do they provide an adequate basis for violation of any other ethical imperatives of this code.

1.5 Honor property rights including copyrights and patents.

Violation of copyrights, patents, trade secrets and the terms of license agreements is prohibited by law in most circumstances. Even when software is not so protected, such violations are contrary to professional behavior. Copies of software should be made only with proper authorization. Unauthorized duplication of materials must not be condoned.

1.6 Give proper credit for intellectual property.

Computing professionals are obligated to protect the integrity of intellectual property. Specifically, one must not take credit for other's ideas or work, even in cases where the work has not been explicitly protected by copyright, patent, etc.

1.7 Respect the privacy of others.

Computing and communication technology enables the collection and exchange of personal information on a scale unprecedented in the history of civilization. Thus there is increased potential for violating the privacy of individuals and groups. It is the responsibility of professionals to maintain the privacy and integrity of data describing individuals. This includes taking precautions to ensure the accuracy of data, as well as protecting it from unauthorized access or accidental disclosure to inappropriate individuals. Furthermore, procedures must be established to allow individuals to review their records and correct inaccuracies.

This imperative implies that only the necessary amount of personal information be collected in a system, that retention and disposal periods for that information be clearly defined and enforced, and that personal information gathered for a specific purpose not be used for other purposes without consent of the individual(s). These principles apply to electronic communications, including electronic mail, and prohibit procedures that capture or monitor electronic user data, including messages,without the permission of users or bona fide authorization related to system operation and maintenance. User data observed during the normal duties of system operation and maintenance must be treated with strictest confidentiality, except in cases where it is evidence for the violation of law, organizational regulations, or this Code. In these cases, the nature or contents of that information must be disclosed only to proper authorities. (See 1.9)

1.8 Honor confidentiality.

The principle of honesty extends to issues of confidentiality of information whenever one has made an explicit promise to honor confidentiality or, implicitly, when private information not directly related to the performance of one's duties becomes available. The ethical concern is to respect all obligations of confidentiality to employers, clients, and users unless discharged from such obligations by requirements of the law or other principles of this Code.

2. MORE SPECIFIC PROFESSIONAL RESPONSIBILITIES.

As an ACM computing professional I will . . .

2.1 Strive to achieve the highest quality, effectiveness and dignity in both the process and products of professional work.

Excellence is perhaps the most important obligation of a professional. The computing professional must strive to achieve quality and to be cognizant of the serious negative consequences that may result from poor quality in a system.

2.2 Acquire and maintain professional competence.

Excellence depends on individuals who take responsibility for acquiring and maintaining professional competence. A professional must participate in setting standards for appropriate levels of competence, and strive to achieve those standards. Upgrading technical knowledge and competence can be achieved in several ways:doing independent study; attending seminars, conferences, or courses; and being involved in professional organizations.

2.3 Know and respect existing laws pertaining to professional work.

ACM members must obey existing local, state,province, national, and international laws unless there is a compelling ethical basis not to do so. Policies and procedures of the organizations in which one participates must also be obeyed.But compliance must be balanced with the recognition that sometimes existing laws and rules may be immoral or inappropriate and,therefore, must be challenged. Violation of a law or regulation may be ethical when that law or rule has inadequate moral basis or when it conflicts with another law judged to be more important. If one decides to violate a law or rule because it is viewed as unethical, or for any other reason, one must fully accept responsibility for one's actions and for the consequences.

2.4 Accept and provide appropriate professional review.

Quality professional work, especially in the computing profession, depends on professional reviewing and critiquing. Whenever appropriate,individual members should seek and utilize peer review as well as provide critical review of the work of others.

2.5 Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks.

Computer professionals must strive to be perceptive, thorough, and objective when evaluating, recommending, and presenting system descriptions and alternatives. Computer professionals are in a position of special trust, and therefore have a special responsibility to provide objective, credible evaluations to employers, clients, users, and the public. When providing evaluations the professional must also identify any relevant conflicts of interest, as stated in imperative 1.3.

As noted in the discussion of principle 1.2 on avoiding harm, any signs of danger from systems must be reported to those who have opportunity and/or responsibility to resolve them. See the guidelines for imperative 1.2 for more details concerning harm,including the reporting of professional violations.

2.6 Honor contracts, agreements, and assigned responsibilities.

Honoring one's commitments is a matter of integrity and honesty.For the computer professional this includes ensuring that system elements perform as intended. Also, when one contracts for work with another party, one has an obligation to keep that party properly informed about progress toward completing that work.

A computing professional has a responsibility to request a change in any assignment that he or she feels cannot be completed as defined. Only after serious consideration and with full disclosure of risks and concerns to the employer or client, should one accept the assignment. The major underlying principle here is the obligation to accept personal accountability for professional work. On some occasions other ethical principles may take greater priority.

A judgment that a specific assignment should not be performed may not be accepted. Having clearly identified one's concerns and reasons for that judgment, but failing to procure a change in that assignment, one may yet be obligated, by contract or by law, to proceed as directed. The computing professional's ethical judgment should be the final guide in deciding whether or not to proceed. Regardless of the decision, one must accept the responsibility for the consequences.

However, performing assignments "against one's own judgment" does not relieve the professional of responsibility for any negative consequences.

2.7 Improve public understanding of computing and its consequences.

Computing professionals have a responsibility to share technical knowledge with the public by encouraging understanding of computing, including the impacts of computer systems and their limitations. This imperative implies an obligation to counter any false views related to computing.

2.8 Access computing and communication resources only when authorized to do so.

Theft or destruction of tangible and electronic property is prohibited by imperative 1.2 - "Avoid harm to others." Trespassing and unauthorized use of a computer or communication system is addressed by this imperative. Trespassing includes accessing communication networks and computer systems, or accounts and/or files associated with those systems, without explicit authorization to do so. Individuals and organizations have the right to restrict access to their systems so long as they do not violate the discrimination principle (see 1.4). No one should enter or use another's computer system, software, or data files without permission. One must always have appropriate approval before using system resources, including communication ports, file space, other system peripherals, and computer time.

3. ORGANIZATIONAL LEADERSHIP IMPERATIVES.
As an ACM member and an organizational leader, I will . . . . .

BACKGROUND NOTE:This section draws extensively from the draft IFIP Code of Ethics,especially its sections on organizational ethics and international concerns. The ethical obligations of organizations tend to be neglected in most codes of professional conduct, perhaps because these codes are written from the perspective of the individual member. This dilemma is addressed by stating these imperatives from the perspective of the organizational leader. In this context"leader" is viewed as any organizational member who has leadership or educational responsibilities. These imperatives generally may apply to organizations as well as their leaders. In this context"organizations" are corporations, government agencies,and other "employers," as well as volunteer professional organizations.

3.1 Articulate social responsibilities of members of an organizational unit and encourage full acceptance of those responsibilities.

Because organizations of all kinds have impacts on the public, they must accept responsibilities to society. Organizational procedures and attitudes oriented toward quality and the welfare of society will reduce harm to members of the public, thereby serving public interest and fulfilling social responsibility. Therefore,organizational leaders must encourage full participation in meeting social responsibilities as well as quality performance.

3.2 Manage personnel and resources to design and build information systems that enhance the quality of working life.

Organizational leaders are responsible for ensuring that computer systems enhance, not degrade, the quality of working life. When implementing a computer system, organizations must consider the personal and professional development, physical safety, and human dignity of all workers. Appropriate human-computer ergonomic standards should be considered in system design and in the workplace.

3.3 Acknowledge and support proper and authorized uses of an organization's computing and communication resources.

Because computer systems can become tools to harm as well as to benefit an organization, the leadership has the responsibility to clearly define appropriate and inappropriate uses of organizational computing resources. While the number and scope of such rules should be minimal, they should be fully enforced when established.

3.4 Ensure that users and those who will be affected by a system have their needs clearly articulated during the assessment and design of requirements; later the system must be validated to meet requirements.

Current system users, potential users and other persons whose lives may be affected by a system must have their needs assessed and incorporated in the statement of requirements. System validation should ensure compliance with those requirements.

3.5 Articulate and support policies that protect the dignity of users and others effected by a computing system.

Designing or implementing systems that deliberately or inadvertently demean individuals or groups is ethically unacceptable. Computer professionals who are in decision making positions should verify that systems are designed and implemented to protect personal privacy and enhance personal dignity.

3.6 Create opportunities for members of the organization to learn the principles and limitations of computer systems.

This complements the imperative on public understanding (2.7). Educational opportunities are essential to facilitate optimal participation of all organizational members. Opportunities must be available to all members to help them improve their knowledge and skills in computing, including courses that familiarize them with the consequences and limitations of particular types of systems.In particular, professionals must be made aware of the dangers of building systems around oversimplified models, the improbability of anticipating and designing for every possible operating condition, and other issues related to the complexity of this profession.

4. COMPLIANCE WITH THE CODE. As an ACM member I will ...

4.1 Uphold and promote the principles of this Code.

The future of the computing profession depends on both technical and ethical excellence. Not only is it important for ACM computing professionals to adhere to the principles expressed in this Code, each member should encourage and support adherence by other members.

4.2 Treat violations of this code as inconsistent with membership in the ACM.

Adherence of professionals to a code of ethics is largely a voluntary matter. However, if a member does not follow this code by engaging in gross misconduct, membership in ACM may be terminated.

This Code and the supplemental Guidelines were developed by the Task Force for the Revision of the ACM Code of Ethics and Professional Conduct: Ronald E. Anderson, Chair, Gerald Engel, Donald Gotterbarn, Grace C. Hertlein, Alex Hoffman, Bruce Jawer, Deborah G. Johnson, Doris K. Lidtke, Joyce Currie Little, Dianne Martin, Donn B. Parker, Judith A. Perrolle, and Richard S. Rosenberg. The Task Force was organized by ACM/SIGCAS and funding was provided by the ACM SIG Discretionary Fund. This Code and the supplemental Guidelines were adopted by the ACM Council on October 16, 1992.

The Four Step Analysis Process *

Step I. Analyze the situation. What is the subject of the case; what is it all about?

A. What are the relevant facts?
B. Who are the stakeholders, that is, who has an interest, or stake, in the outcome?

Step II. Make a defensible ethical decision? (Refer to Chapter 1 for details.)

A. Isolate the ethical issues.
  1. Should someone have done or not done something?
  2. Does it matter that ...? (reasons and excuses)
B. Examine the legal issues.
C. Consult guidelines.
  1. Do corporate policies apply?
  2. What codes of conduct apply?
  3. Does the action violate the Golden Rule?
  4. Who benefits? Who is harmed?
  5. Does the action pass tests for right and wrong?
D Discover the applicable ethical principles.
  1. Explore ways to minimize harm.
  2. Analyze pertinent rights and duties.
  3. Define professional responsibilities.
  4. Examine the situation in terms of egoism and utilitarianism.
  5. Apply concepts of consistency and respect.
E. Make a defensible choice.

STEP III. Describe steps to resolve the current situation.

A. What are the options at this time?
B. Which option(s) do you recommend?
C. Defend the legality and ethicality of your recommendation.
D. How would you implement your recommendation?
E. Recommend short-term corrective measures.
  1. Analyze the pivot points.
  2. Alter the parameters.

STEP IV. Prepare policies and strategies to prevent recurrence.

A. What organizational, political, legal, technological, societal changes are needed?
B. What are the consequences of your suggested changes?
  1. What happens when this resolution is invoked?
  2. What obstacles might prevent your plan from working?
  3. Why should the organization implement the changes?
  4. How do the changes benefit the organization? Are they marketable, or do they further public relations? (Perhaps perform a cost/benefit analysis.)
  5. Do the changes increase the net good for those concerned? Does anyone get hurt?
  6. Do the changes reflect human rights and reflect common duties?

* Source: Ethical Decision Making and Information Technology by Kallman and Grillo (Mitchell McGraw-Hill, 1993)

CASE WORKSHEET (see Chapter 3 for details on how to carry out each step) *

I. Find the facts
   A. List the relevant facts:

   B. List the stakeholders:

II. Make a defensible ethical decision.
   A. Isolate ethical issues (Should someone have/have not done something?)

   B. Examine the legal issues:

   C. Consult guidelines

   Corporate policies, codes of Conduct:

   Golden Rule

   Who benefits? Who is harmed?

   Tests for right and wrong:

D. Discover the applicable ethical principles

Least harm

Rights and duties

Professional responsibilities

Self interest and utilitarianism

Consistency and respect

E. Make a defensible choice

III. Describe steps to resolve the current situation.

A. Options

B. Recommendation

C. Defense

D. Implementation

E. Short-term corrective measures

IV. Prepare policies and strategies to prevent recurrence

A. Describe the organizational, political, legal, technological, or societal changes needed:

B. Describe the consequences of your suggested changes:

Real-World Software Engineering

IV PROJECTS

This section is divided into five parts. Part a provides an overview of the three types of projects (small, extended, maintenance) around which the course is organized. This is followed by a discussion about significant issues to consider when selecting projects for a project oriented class (part b). Once a project is chosen it must be carefully managed and evaluated. Project management techniques and tools are presented in part c. Sample projects are included in part d. Part e consists of an extended discussion of our approach to the management of a large project.

a. Introduction to Projects.

The course is organized around three types of project. A brief description of each of the three projects follows.

1. Small project - The requirements are provided to the students who are expected to specify, design, code, and test a solution. The small project is scheduled for weeks 2 through 6 of the first semester. Since work must begin quickly, controlling disciplines are imposed upon the teams with minimum justification at this point. For example, students are immediately introduced to estimation, scheduling, project organization, configuration management, quality assurance, and verification and validation techniques by "living them" but only later are these topics formally addressed in lectures. While the project is implemented in a language with which the students are already proficient, Ada specifications are used in high-level design.

2. Extended project - Beginning with an initial request from a "real customer", students are expected to complete all aspects of a solution, from requirements engineering (elicitation, analysis, and specification) through implementation. This project begins in week 5 of the first semester and continues through week 11 of the second semester. Analysis and design, through Ada specifications, are to be completed by the end of the first semester with detailed design, coding, and testing to follow in the second semester. Ada is used as the notation for requirements specification and design. Structured analysis and design techniques are applied to this project. DOD-STD-2167a [2167A] is used as the standard for the required documents and procedures. Internal project reviews are emphasized [Bruegge 91].

3. Maintenance project - Students perform major maintenance (including corrective, enhancement, and adaptive activities) on an existing software system. A complete and sound set of system documents, which adhere to accepted software engineering standards, is provided. The maintenance

project is scheduled for weeks 6 through 11 of the second semester. The project chosen is one that has been previously implemented in Ada with object-oriented techniques. A variety of maintenance tasks, including those described by Engle, Ford, and Korson [Engle 89], are assigned. The students are required to perform multiple concurrent maintenance tasks on this large Ada artifact [Callis 91]. These maintenance activities require the use of disciplined change control and the ability to work with a large unfamiliar artifact.

We specifically recommend that students work in project teams, assigned to developing a software product for a customer. The customer may be a real customer who has a genuine software need, or the customer may be role-played by the instructor or a colleague with a simulated software need. In either case, the customer makes a request to initiate the project. The request may be given orally or in writing and consists of a brief statement intended to convey, in the customer's language, the essence of the desired product. At this point the students are charged with developing a problem definition. It is emphasized that defining the problem requires a real understanding of the problem domain. The specific form of the problem definition to be developed will vary depending on the particular course and instructor preference but should be clearly stated and demonstrated by the instructor. Students should also be provided with techniques for eliciting requirements as well as for working in groups. Techniques include interviewing, observation, context-free questions, brainstorming, scenarios, reviews, and effective meetings (Gause & Weinberg, 1989, Pressman, 1988, Weinberg, 1982, Metzger, 1987).

b. Selecting a project

The success of our approach is dependent on the selection of appropriate software projects. The selected software project must accomplish something more than a mere calculation and represent a real world problem of sufficient complexity so that the requirements gathering challenges the students. Other factors in project selection include required domain knowledge, non-functional requirements, and incremental development.

In the real world, the customer has significantly more domain knowledge than the software developer, who must acquire some of that knowledge to successfully develop the system. There is rarely sufficient time within the duration of a class to gain this knowledge. Consequently, both the instructor and the students must already be familiar with the knowledge domain of the selected project. Undertaking a project which requires exotic domain knowledge by the instructor or the students merely adds distracting complications to the project as either struggles to understand the domain. While the choice of an exotic application might excite the students, the instructor may be unable to comfortably portray a knowledgeable customer. Even with familiar tasks, students still have difficulty capturing the customer's needs and designing a solution.

This experience helps them understand the essential difficulty of requirements gathering. Familiarity with the project domain also reduces the need for the instructor to correct unnecessarily complicated design assumptions.

Most students are familiar with common mechanical devices. Computerizing simple mechanical devices is a useful source of project ideas. From one model of a device, many projects can be developed. For example, students understand the model of a machine which accepts money and dispenses products. Variations on this model which we have used on recent projects include the computerization of a snack vending machine and a videotape vending machine. The model can also be reversed to that of a machine which accepts products and dispenses money. A variation on this is a recycling machine which accepts empty cans and bottles and dispenses money in exchange for these items. The similarity of these variations is useful if multiple student projects need to be developed concurrently. Even though students notice this similarity, they quickly learn that each project is very different because of the customer's needs. For example, the snack vending machine is continually replenished by the owner while the videotape vending machine is restocked by a user returning a videotape.

Another factor in project selection is the presence of non-functional requirements. Non-functional requirements are requirements that do not relate directly to the functions or operations to be performed by the system. Students must learn to recognize non-functional requirements such as the need for rapid responses to customer requests and appropriate user interfaces. Again, familiarity with the selected model helps the students to easily identify the non-functional requirements.

The ability to provide incremental development is another factor in project selection. For student satisfaction with the learning experience, it is important for them to deliver a functioning system. If a project has several distinct functional components, it is more likely that at least some of those components can be delivered. For example, a video-tape vending machine keeps track of its tapes, records charges to user's charge cards, accepts money, and accepts and dispenses tapes. Each of these functions can be developed and tested separately, giving the student some sense of accomplishment. Selecting a project conducive to incremental development models the real world in that customers sometimes have to cut back on their expectations. Also, software reuse and integration testing are more easily demonstrated in systems that can be incrementally developed.

In project selection, two strong motivating factors for students have been their familiarity with the problem domain and the possibility of others using their software. We have found that school-related projects are well received by students. Automating aspects of a departmental library or advisement system, tracking the distribution of athletic equipment, automating a dormitory assignment lottery, and detecting program plagiarism are examples of such projects. An additional virtue of school-related

TEST PROCEDURE FORM

Test: E (Total Percentage)

Test Version: 1.0

Description: This test is performed to ensure that the system will advise the user to continue on with the next filter even if the last filter's percentage was not 80% or greater if the total percentage is greater than 80%.

Requirements: 19

Prerequisites: B8

Test Data Required: TD_102A.TST and TD_102B.TST

Test Steps:

1. Enter program names TD_.TST and TD_.TST.
2. Run filter one (should contribute 10% to Total).
3. Run filter two (should contribute 15% to Total).
4. Run filter three (should contribute 20% to Total).
5. Pick each procedure from first program and run with every other procedure from second program in filter four (should contribute 10% to Total).
6. Pick each procedure from first program and run with every other procedure from second program in filter five. (should contribute 10% to Total).
7. Run filter six (should contribute 11.85% to Total because of possible 15% is 11.85%).
8. Run filter seven (should contribute 7.9%).
9. Total shoud be (84.75%) verify system advises you to continue on with filter eight even though filter seven's percentage was only 79%.

RESULTS OF TEST E

EXPECTED	ACTUAL
System Advises to continue with Filter Eight.	

TEST PROCEDURE FORM

Test: F (Menu Key torture test)

Test Version: 1.0

Description: This test is performed to check that Menus do not take incorrect actions when meaningless keys are pressed.

Requirements: 3

Prerequisites: A1

Test Data Required: TD_1A.TST and TD_1B.TST

Test steps:

1. Enter TD_1A.TST for first program.

2. Enter TD_1B.TST for second program.

3. When menu for Filter 1 is displayed press the key indicated in the matrix below and record the results.

4. Exit system.

TEST DATA MATRIX (F)

Press Keys	Expected Result	Actual Result	Pass/ Fail	Comments
<Right Arrow> <RETURN>	Rejected			
<Down Arrow> <RETURN>	Rejected			Customer says this is not a problem do not run.
<F1> <RETURN>	Rejected			
<Gray +> <RETURN>	Rejected			
<RETURN>	Rejected			Customer says this is not a problem do not run.
<[> <RETURN>	Rejected			
<K> <RETURN>	Rejected			
<SPACE BAR> <RETURN>	Rejected			Customer says this is not a problem do not run.
<9> <RETURN>	Rejected			

APPENDIX

Third Eye Plagiarism Detection System Ver 1.0 Beta _____

Test Name	Description	Test Ver	Run by	Problem Report Numbers	Result P=Pass F=Fail N=Not Run
A1	Prompt for source Pgms				
A2	Exit test				
A3	Report information				
B1a	Filter 1 Physical lines / comments				
B1b	Filter 1 constructs				
B1c	Filter 1 percentage				
B1d	Filter 1 torture				
B2a	Filter 2 Globa VARs				
B2b	Filter 2 Global CONSTs				
B2c	Filter 2 Global TYPEs				
B2d	Filter 2 Global functions/procedures				
B2e	Filter 2 Total # of Globals				
B2f	Filter 2 percentage				
B2g	Filter 2 torture				
B3a	Filter 3 function / procedure interfaces				
B3b	Filter 3 percentage				
B3c	Filter 3 torture				
B4a	Filter 4 physical lines / comments				
B4b	Filter 4 constructs				
B4c	Filter 4 percentage				
B4d	Filter 4 torture				

Third Eye Plagiarism Detection System Ver 1.0 Beta _____

Test Name	Description	Test Ver	Run by	Problem Report Numbers	Result P=Pass F=Fail N=Not Run
B5a	Filter 5 Global VARs				
B5b	Filter 5 Global CONSTs				
B5c	Filter 5 Global TYPEs				
B5d	Filter 5 function / procedures				
B5e	Filter 5 total # of Globals				
B5f	Filter 5 percentage				
B5g	Filter 5 torture				
B6a	Filter 6 keywords				
B6b	Filter 6 percentage				
B7a	Filter 7 Identifiers				
B7b	Filter 7 functions / procedures				
B7c	Filter 7 percentage				
B7d	Filter 7 torture				
B8a	Filter 8 functions				
B8b	Filter 8 procedures				
B8c	Filter 8 percentage				
B8d	Filter 8 torture				
C	30 seconds/response				
D	Help test				
E	Total percentage test				
F	Menu Key torture test				

Problem Report Form

PROBLEM SUBMISSION: Problem Report #_____
Originator:
Date:
Version Number Tested:

Problem Description:

Was the problem found by a test? Yes__ No__
If yes, give test name:

Input:

Expected Result:

Actual Result:

Additional Comments:

===
PROBLEM RESOLUTION:
Name:
Date:
Disposition

Problem Fixed ___ Not a Problem ___ Duplicate Problem ___

If this is a duplicate problem then give the number of the report on which this problem was previously identified ___

Comments:

Problem Report Tracking Form

Report Number	Ver Tested	Date Reported	Date Returned	Disposition: Fixed, Not a Problem, Duplicate	Closed

Sequence Of Test Execution Form

Tester Name ______________________

Version Tested ____________________

Test Order	Test Name	Date	Time	Comments

VI Student assessment

We use tests and quizzes for two reasons; to encourage students to keep up on their reading assignments and to evaluate their understanding of fundamental software engineering concepts. Students are encouraged to keep up with the reading assignments by the use of frequent quizzes. Students who have completed their reading assignments before the quiz, will generally do well on them. An in-class review of the quiz enables the instructor to locate confusions about the reading material in real-time. The discovery of confusion is not delayed until a major test has been given and graded. Concepts are both learned and lived. The major written examinations go over the details of the readings. By the time a major examination is given, students have read the material, had a quiz on it, and started to apply the principles in a project.

Below are several sample quizzes and the answers we would accept. There are also some sample tests. We give several types of tests. Some tests have both a take home and an in-class part. The take-home portion is used for questions which require careful application of concepts learned in class and questions on professional issues.ake home portion of the exam. We also have students review deliverables as take home portions of the exam. This motivates them to carefully synthesize and apply concepts learned in class. A composite of their criticisms of the first semester deliverables is used when introducing the extended project at the beginning of the second semester. The in-class portion of each test is preceded by a study guide. The study guide is included here along with each test.

QUIZZES

CSCI 3250 QUIZ/EXERCISE 1 **NAME:**

1. The qualities looked for in software depend on one's point of view. Mynatt describes three different categories of people to whom good software quality is important. Each views a software system from a different perspective.

   Name the three categories of people and, for each, list one software quality that is important to them.

---

a. Sponsor - Any of following qualities
- **Low costs**
- **Increased productivity**
- **Flexibility**
- **Efficiency**
- **Reliability**

b. User - Any of following qualities
- **Functionality**
- **Ease of learning**
- **Ease of remembering**
- **Ease of use**
- **Efficiency**
- **Reliability**

c. Maintainer/modifier - Any of following qualities
- **Minimum errors**
- **Good documentation**
- **Readable code**
- **Good design**
- **Reliability**

CSCI 3250 QUIZ/EXERCISE 2 **NAME:**

Choose one, <u>but not both</u>, of the questions below and answer it.

---

1. According to Mynatt, the <u>analysis</u> phase actually has two components. Name and briefly describe these two parts of the analysis phase.

   (1) <u>Requirements analysis</u> involves defining the problem and the requirements that must be met.

   (2) <u>Specification</u> describes the technical requirements for the system; e.g. specifying what the system is to do and any operational constraints. Specification is a refinement of requirements analysis and involves the creation of testable requirements.

---

2. Structured analysis is a widely used approach to analysis. Name and briefly describe the components, according to Mynatt, of a structured analysis model.

   Context diagram

   Data flow diagrams

   Data dictionary

   Activity specifications

CSCI 3250 QUIZ/EXERCISE 3 **NAME:**

Answer <u>only one</u> of the three questions that follow.

1. There are two software design strategies that have been widely adopted. Name and briefly describe each.

2. According to Sommerville, "the most important design quality attribute is maintainability." Coupling and cohesion affect the maintainability of a system. Briefly define and distinguish between coupling and cohesion.

3. Are structure charts analysis or design documents? What does a structure chart show?

CSCI 3250 QUIZ/EXERCISE 4 **NAME: Answer Key**

Answer <u>only one</u> of the two questions that follow.

1. (a) What is black-box testing? (b) What is white-box testing? (c) For black-box testing, from what are the tests derived (i.e. what must you have knowledge of in order to construct black-box tests)? (d) Repeat part (c) for white-box testing.

    (a) **Black-box testing, or functional testing, tests the system from an external standpoint; i.e. the system is viewed as a black-box whose behavior is examined solely from its inputs and related outputs..**
    (b) **White-box testing, or structural testing, considers the internals of the system; i.e. the internal structure (the code) is examined to consider what tests to run. Tests are constructed so as to "cover" (to exercise) all of the code.**
    (c) **Black-box tests are derived from the system's specifications.**
    (d) **White-box tests are derived from the internal program structure; i.e. the code.**

2. Name and describe the three categories of software maintenance. (b) Which type is the most prevalent (i.e., in practice, which category comprises the highest percentage of software maintenance efforts)?

    (a) **Corrective - maintenance to correct defects**

    Perfective (or enhancement) - maintenance to add new features and functionality; i.e. due to a change in requirements

    Adaptive - maintenance due to a change in the operational ervironment

    (b) **Perfective is the most prevalent; approximately 65% of all maintenance is perfective.**

CSCI 3250 QUIZ/EXERCISE 5 **NAME:**

1. Distinguish between process documentation and product documentation and give an example of each.

Process documentation deals with the software development process. Process documentation includes such things as project management plans, schedules, development standards, and meeting notes/reports. Its primary purpose is to assist in the management of software development.

Product documentation deals with the software product that is being developed. It includes such things as users manuals, installation guides, requirements documents, design documents, source code, and test plans.

CSCI 3250 QUIZ/EXERCISE 6 **NAME: Answer Key**

(a) Distinguish between functional requirements and non-functional requirements.
(b) For your team's project, give a specific example of a functional requirement and of a non-functional requirement.

(a) Functional requirements are requirements that relate directly to the system's functionality; i.e., they describe how the system should behave.

Non-functional requirements do not relate to functions of the system. They are requirements or constraints (restrictions) under which the system must operater and standards that must be met.

(b) See Mynatt, pages 70-74, and Sommerville, pages 92-95, for ideas.

CSCI 3250 QUIZ/EXERCISE 7 **NAME:**

Answer only one of the two questions that follow.

---

1. (a) According to Sommerville, the reliability of a software system is a measure of:

how well the software provides the services expected of it

by its users.

(b) Sommerville discusses four different metrics which have been used to assess software reliability. Name two of these reliability metrics.

Any two of the following:

(1) **Probability of failure on demand** (3) **Mean time to failure**

(2) **Rate of failure occurrence** (4) **Availability**

---

2. Mynatt describes five general factors that are related to user interface quality. Identify four of them.

Any four of the following:

(1) **Ease of learning** (4) **User satisfaction**

(2) **Speed of use** (5) **Knowledge retention**

(3) **Frequency of user errors**

CSCI 3250 QUIZ/EXERCISE 8 **NAME: Composite of student responses**

All serious responses will be considered correct!

There has been a lot of discussion about ethics and computing. List what you think are three ethical issues in computing.

* Security of data being transmitted electronically (encryption/decryption)
* Software ownership; licensing; piracy; plagiarism; pricing
* Accessing information without authorization
* Computer crime
* Software quality:
  - released without adequate testing or with known deficiencies
  - providing quality product at reasonable cost
  - safety critical systems
* Support policies (on <u>outside</u> of package)
* Viruses
* Privacy: - on systems such as Compuserve
* Issue of employee developing software for one employer and not be allowed to take it to anouther employer
* AI - trying to duplicate human brain, human thoughts
* Monitoring: - e-mail
* Abuse of resources on Internet
* Copying processor architecture

CSCI 3250 QUIZ/EXERCISE 9 **NAME: Answer Key**

1. Complete the following sentence in 5 words or less:

   Configuration management is concerned with change control.

2. What does it mean when something is baselined?

   It is placed under configuration management control (i.e. it becomes a configuration item) and thereafter changes can only be made by going through the established change control process.

3. One of the most important roles of quality assurance (QA) is the development of product standards and process standards.

   (a) Give a specific example of a process standard.

Any of the following:	**Design review conduct**
	Submission of documents to CM
	Version release process
	Project plan approval process
	Change control process
	Test recording process

   (b) Give a specific example of a product standard.

Any of the following:	**Design review form**
	Document naming standard
	Procedure header format
	Programming standards
	Project plan format
	Change request form

EXAMINATIONS

TEST 1 STUDY GUIDE

Note:
1. Test 1 covers lectures and reading through week 7 of the syllabus.
2. Understand the concepts and be able to apply them.
3. If a topic is not on review sheet, it is not on the test.

1. Name the phases of the software life cycle. For each, describe the purpose, activities carried out in the phase, inputs, outputs, and documents produced.

2. Understand the components of the structured analysis and design models being used in your team project, including the notation used. This includes:
   a) statement of scope (narrative overview); b) context diagram;
   c) data flow diagrams (leveling, balancing); d) data dictionary;
   e) structure chart; f) module descriptions

3. Sommerville says that a major difficulty is establishing large software system requirements is that the problems are usually "wicked problems." What is a wicked problem and what is the major difficulty? Give some examples?

4. Distinguish between a requirements definition and a requirements specification? Why is it useful to draw a distinction, as Sommerville does, between a requirements definition and a requirements specification?

5. a) Distinguish between functional requirements and non-functional requirements? Give examples.
   b) Describe the three principal classes of non-functional requirements.

6. Consider Exercises 5.2 and 5.8 (page 100) of Sommerville.

7. Regarding software reviews, or walkthroughs:
   a) What is the purpose?
   b) Who participates and what are the different roles?
   c) What are some standard guidelines for effective reviews?

8. Define and distinguish between coupling and cohesion? Explain why maximizing cohesion and minimizing coupling leads to more maintainable systems.

9. In evaluating a design, two major issues need to be considered. First, does the design fulfill the requirements; second, does it meet established design

standards? What types of standards are being referred to?

10. What are the three key design goals?

11. Describe these process models and give the strengths and weaknesses of each.
a) waterfall model; b) prototyping; c) spiral model.
d) Consider Exercise 6.7, page 119, of Sommerville.

12. Qualities looked for in software depend on one's point of view. Mynatt describes three categories of people to whom good software quality is important. Each views a software system from a different perspective. Name the three categories of people and, for each, list the software qualities that are important to them.

13. According to Sommerville, what are the key attributes of a well-engineered software system, assuming it provides the required functionality?

14. According to Mynatt, the analysis phase actually has two components. Name and briefly describe these two parts of the analysis phase.

15. Name and describe the two widely adopted software design strategies.

16. a) What is black-box testing? Describe some black-box testing techniques.
b) What is white-box testing? Describe some white-box testing techniques.
c) For black-box testing, from what are the tests derived (i.e. what must you have knowledge of in order to construct black-box tests)?
d) For white-box testing, from what are the tests derived (i.e. what must you have knowledge of in order to construct white-box tests)?

17. Name,describe, and give specific examples of the three types of software maintenance. Which is the most prevalent? Which would you expect to be most prevalent in the first six months after a new product is released. Why?

18. Distinguish between process documentation and product documentation and give examples of each. For each type, for whom is it intended, what is the purpose, and how long will the documentation likely be used?

19. Discuss factors which affect group communication. Give suggestions that an organization could consider in improving group effectiveness.

20. What is meant by traceability of requirements throughout the software

development process. Describe specifically how this could be provided in design? in testing?

CSCI 3250 TEST 1, NAME: **Answer Key**

1. (32 points) Write the letter of the correct answer in the space provided to the left of each question.

<u>B</u> A. The phase in which the software architecture is established is
(a) requirements analysis & specification (d) maintenance
(b) design (e) all of these
(c) implementation

<u>A/B</u> B. The phase in which test plans and test cases are developed is
(a) requirements analysis & specification (d) maintenance
(b) design (e) all of these
(c) implementation

<u>E</u> C. The phase in which documentation is produced is
(a) requirements analysis & specification (d) maintenance
(b) design (e) all of these
(c) implementation

<u>A</u> D. The phase in which the client's problem is defined and recorded is
(a) requirements analysis & specification (d) maintenance
(b) design (e) all of these
(c) implementation

<u>B</u> E. The phase in which the structure chart is developed is
(a) requirements analysis & specification (d) maintenance
(b) design (e) all of these
(c) implementation

* <u>A</u> F. The phase in which the leveled set of data flow diagrams is developed is
(a) requirements analysis & specification (d) maintenance
(b) design (e) all of these
(c) implementation

<u>C</u> G. The phase in which module testing occurs is
(a) requirements analysis & specification (d) maintenance
(b) design (e) all of these
(c) implementation

_C_ H. The components of a well designed software system will have a ______ level of cohesion and a ______ level of coupling.
(a) low, low (b) low, high (c) high, low (d) high, high

_E_ I. A structure chart does not show
(a) interfaces between modules
(b) the modules in the system
(c) module hierarchy (who calls who)
(d) functions of modules
(e) execution order
(f) all these are shown

* _A_ J. Black box testing often reveals errors in the interfaces between modules.
(a) true (b) false

_A_ K. Traditionally, software maintenance activities consume more resources than software development activities.
(a) True (b) False

_B_ L. A structured analysis model of a software system (context diagram, data flow diagrams, process specifications, data dictionary) is an example of
(a) process documentation
(b) product documentation
(c) both (a) and (b)
(d) neither (a) nor (b)

_B_ M. The users manual for a system is an example of
(a) process documentation
(b) product documentation
(c) both (a) and (b)
(d) neither (a) nor (b)

_B_ N. The purpose of a review is to identify and correct errors in the reviewed item.
(a) True (b) False

* _A_ O. In order to determine whether a child data flow diagram is balanced with respect to its parent data flow diagram, one would need to look in the data dictionary.
(a) True (b) False

_B_ P. Are the Gazebo Lottery, Kiosk Vending Machine, and Pavilion Recycling Systems "wicked problems"?
(a) Yes (b) No

2. (4 points) Sommerville says "assuming the software provides the required functionality, there are four key attributes which a well-engineered software system should possess". What are they?

1) Maintainable	**3) Efficient**
2) Reliable	**4) Appropriate user interface**

3. (8 points) (a) Name and distinguish between the three categories of software maintenance. Use examples to make sure the distinction is very clear.

   1) Corrective - correcting defects (EXAMPLES)
   2) Perfective (Enhancement) - adding new feature (change in requirements), usually at users request
   3) Adaptive - modification due to change in operational environment of system

   (b) Which type of maintenance would you expect to be most prevalent in the first few months following the release of a system? **Corrective**
   (c) Which type of maintenance would you expect to be most prevalent after the system has been in operation for a couple of years? **Perfective (enhancement)**

4. (6 points) (a) Name the three key software design goals.
   Maintainability, testability, reuseability

   (b) Explain how coupling and cohesion relates to each of these design goals.
   Design consisting of highly cohesive, loosely coupled components is more maintainable, more easily tested, with greater potential for reuse. Why?
   Cohesive modules perform a single task; loosely coupled modules have minimal dependencies with other modules. Therefore when maintenance is needed, the number of modules affected is minimized and thus the chance of an error/change in one module affecting other modules is lowered. Cohesive modules tend to be more general and perform a single task and thus have a higher probability for reuse.

5. (5 points) Regarding software life cycle models:
   (a) The spiral model is sometimes characterized as a "risk-driven" model. Explain.

   A key characteristic is the assessment of risk at regular stages in the project and the initiation of actions to address the risks.
   Risk analysis (identifying and addressing them) is done before each cycle; review procedures to assess the risk is done at the end of each cycle.

(b) In general, for what type of system would the waterfall model be most appropriate? the spiral model?

Waterfall - requirements stable, well defined; i.e. specification risk is low, little/no need for risk resolution.; data processing applications

Spiral - high risk systems; requirements are difficult to pin down; unstable (Schach) internal projects; large projects

6. (8 points) (a) Classify each of the following as **F** (functional) or **N** (non-functional) requirements. For any non-functional requirement, indicate which of the three types of non-functional requirement it is.

__N__ All requirements specifications documents must conform to DOD-2167A.
Process
__F__ The system must retrieve any customer account given customer id or name.
__F__ The system must display any customer account on demand.

(b) Improve the following requirements statements so that they are testable.

The HVAC monitoring system must significantly reduce electrical consumption throughout the organization.

The HVAC monitoring system must reduce electrical consumption by a minimum of 15% throughout the organization.

The system should be easy for experienced controllers to use and should be organized so that user errors are minimized.

Experienced controllers should be able to use all of the system functions after a total of two hours training.

After this training, the average number of errors made by experienced useres should not exceed two per day. user errors are minimized.

7. (6 points) (a) Distinguish between white box and black box testing, including the purpose of each and the relationship between the two.

   Black-box - functional testing; looks at system externally (how does it behave to given inputs); tests derived from specifications

   White-box - structural testing; tests rely on knowledge of internal structure

   Relationship - Both are necessary; neither sufficient

   (b) Briefly name and describe one black box method.

   Equivalence partitioning - Classes (partitions) of input data with common properties (system should respond in same way for all members of the class) are identified and used to derive test cases. System tested with a test from each partition.

8. (5 points) It is desirable in a software development project to provide traceability of requirements throughout the development process.
   (a) In the deliverables required of the Gazebo, Kiosk, and Pavilion project teams, was traceability between requirements and test plan documents required? Explain.

   Yes - refer to required deliverables

   (b) In the deliverables required of the Gazebo, Kiosk, and Pavilion project teams, was traceability between requirements and design documents required? Explain.

   No - refer to required deliverables; while this is desirable it was not required in your deliverables.

9. (6 points) Explain the meaning of the following data dictionary entries.
   (a) digit = 1|2|3|4|5|6|7|8|9|0
   payment = '\$' + $\{\text{digit}\}^4_0$ [+ '.' + digit + digit]

   Payment of any of following forms: (\$, 0-4 digits, .xx optional)

\$9999.99	**\$9999**
\$999.99	**\$999**
\$99.99	**\$99**
\$9.99	**\$9**
\$.99	**\$**

(b) payment_method = cash|check|money_order|charge_card

Payment method is cash or check or money order or charge card

(c) CourseRequestForm = Name + Semester + {Course ID + Hrs}
+ [Dean signature * overload*]
Semester = Fall | Spring | Summer

Course request form: name
semester (Fall or Spring or Summer)
iteration of Course ID and Hrs
optional Dean's signature

Show picture of it

10. (20 points) Consider the following problem specification for a car registration system.

A taxpayer in the Hawaii must register his/her car in order to obtain a license plate. To register a car, the owner must present a proof of ownership, proof of insurance, and payment for the license plate. After registration material is completed, the owner is provided with a car registration and a license plate.

The Hawaii highway patrol (HHP) also makes requests to the car registration system. The HHP can request the owner of a particular registration number or the registration number(s) belonging to a particular owner.

(a) Draw the context diagram.
(b) Draw the first-level data flow diagram.

See attached diagrams

(a) Consider notation & general guidelines

(b) Consider notation & general guidelines
Consider balancing with context diagram
Separate registration and HHP services
Separate validation of registration info & producing it
Separate HHP determining owner given registration & vice versa

COURSE REQUEST FORM

NAME: ______________________________ SEMESTER: ___ Fall
___ Spring
___ Summer

COURSE ID	HRS

____
Dean's Signature (for course overload)

CSCI 3250-201: SOFTWARE ENGINEERING

TEST 1, PART 1 - TAKE-HOME

The following questions comprise one-half of Test 1. Your answers are due at the start of class on Thursday, March 4.

Use a word processor where appropriate. Answer in complete sentences and paragraphs, double-spaced. 25% of your grade for this portion of the test will be based on your writing.

The questions cover both assigned reading and in-class lectures and discussion. Answer in your own words. Use direct quotes sparingly and always indicate when you are quoting another source and cite the source. When in doubt, refer to the AS&T *Language Skills Handbook* for clarification.

You are to work on this test individually. While discussing the questions with others to clarify them is permissible, solutions are to be done individually.

1. Explain how both the waterfall model of the software process and the prototyping model can be accommodated in the spiral process model.

2. Describe what is meant by the terms cohesion, coupling, and adaptability as design criteria. Explain why maximizing cohesion and minimizing coupling leads to more maintainable systems.

3. Develop a class object model, using the notation described in class, for the *Recycling Project Description* attached.

4. The design of a spelling checker is discussed in Appendix A of Sommerville (section 4, pages 617-619) to illustrate design description languages. Document this design by producing:
   (a) a context diagram;
   (b) a leveled and balanced set of data flow diagrams;
   (c) a data dictionary; and
   (d) a structure chart.

RECYCLING PROJECT DESCRIPTION

A system is needed to control a recycling machine for returnable glass bottles, plastic bottles and metal cans. The machine can be used by up to three customers at the same time and each customer can return all three types of items. These items come in various types and sizes. The machine must check which type of item was turned in so that it can print a receipt. A receipt, which can be taken to a cashier, will be printed out. The total value of the items turned in will be printed on one line and the value of each item type will be printed on separate types for each line.

The machine has to be maintained so there is information for the maintenance operator which consists of the total quantity of each item type that has been turned in since the last time the totals were cleared. This information should be able to be printed out. In addition to these totals, the maintenance operator should be able to change the values assigned to individual item types. The machine has numerous mechanical functions which can go awry. The machine has an alarm which indicates that an item is stuck or that the receipt roll is out of paper.

To return items the customer first presses the receipt button to clear all totals. The system then places the items into the correct item type slots. With each item deposited the machine increases the daily totals and the customer totals for that item type. The customer presses the receipt button again to indicate the end of his transaction. The action prints the receipt and updates the daily totals.

The operator needs the ability to turn the alarm off, print the daily reports, and clear the report totals. Not only can the value of the items be changed, but because manufacturers regularly change their packaging, the operator must be able to change the allowable sizes for each item type. When items are stuck the customer is prevented from inserting more items but that customers totals are not lost. After the stuck item is cleared from the machine, the customer can continue to insert items which are added to his/her previous totals.

CSCI 3250-201: SOFTWARE ENGINEERING
STUDY GUIDE FOR TEST 1, PART 2 - IN-CLASS

1. Sommerville says "assuming the software provides the required functionality, there are four key attributes which a well-engineered software system should possess". What are they?

2. Name and briefly describe/compare the three types of software maintenance activities.

3. Familiarize yourself with the requirements of the attached Client Request: Small College Registration System.

4. What are the fundamental differences between object-oriented and function-oriented design?

5. Regarding verification and validation:
   (a) What is the difference between verification and validation and why is validation a particularly difficult process?
   (b) What is the difference between testing and debugging?
   (c) Define the following types of testing: unit, module, sub-system, system, acceptance, alpha, beta, regression.
   (d) Understand these testing strategies: top-down, bottom-up, thread, stress.
   (e) Are V&V and Software Quality Assurance synonyms? Explain.

6. Understand these object-oriented terms: class, object (state, behavior, identity), abstraction, encapsulation, modularity, hierarchy (2 types), inheritance, polymorphism.

7. Transform analysis is a strategy for obtaining a first-cut structure chart from a data flow diagram. One step in transform analysis is identifying the central transform. What is the central transform and how can it be identified?

8. Regarding configuration management (CM):
   (a) What is CM and what are the four major CM activities?
   (b) What happens when something is placed under configuration control (i.e. becomes a configuration item)?
   (c) What are the configuration items in a well-engineered software project?

9. According to Sommerville, would a good software development organization have a Software Quality Assurance (SQA) Group responsible for SQA throughout the organization or various SQA teams, one associated with each software project? Do you agree with Sommerville? Can you think of an advantage and disadvantage of each type of SQA organization?

CLIENT REQUEST: SMALL COLLEGE REGISTRATION SYSTEM

The Dean of the Faculty at a small college wants to automate their student registration system. Following is her request.

We want to retain many of the features of our current manual registration system while automating some of the paperwork. Specifically, we want to retain the interaction between faculty and students during the registration process. We visualize the following process.

Each faculty member will continue to participate in registration. Each will sit at a table in the student center with a set of "slot cards" for each course they teach. They'll have one slot card for each position in a class (i.e. if the maximum enrollment for a class is 12, then there will be 12 slot cards for that class). A slot card contains the course number, location, time, and space for student name and instructor's signature. Each faculty member also has a blank class roster for each course containing course number, location, time, and spaces for names of students in the class.

A schedule booklet is sent to each student approximately two weeks prior to registration. At the same time each student is sent an up-to-date transcript.

On registration day, students go to the student center and pick up a slot card for each course they want to take. Before giving them a slot card, the faculty member looks at their transcript (to check prerequisites, etc) and takes one of the following actions:

(a) completes a slot card for the course (adds student name and faculty signature), adds the students name to the class roster, and gives the slot card to the student; or
(b) tells the student they can't take the course.

When finished, students turn in their slot cards and instructors turn in their class rosters. The (new) automated system is then used to officially register students. The system generates a schedule and bill for each student, an official class roster for each instructor, and a "discrepancy report" for each student or instructor, as appropriate. Discrepancies occur in two ways: first, whenever a slot card exists for a given student but that student does not appear on the class roster submitted by the instructor; and second, whenever a student does appear on the class roster submitted by an instructor but a corresponding slot card is not submitted.

CSCI 3250-201: SOFTWARE ENGINEERING

TEST 1, PART 1 - TAKE-HOME

The following questions comprise one-half of Test 1. Your answers are due at the start of class on Thursday, March 4.

Use a word processor where appropriate. Answer in complete sentences and paragraphs, double-spaced. 25% of your grade for this portion of the test will be based on your writing.

The questions cover both assigned reading and in-class lectures and discussion. Answer in your own words. Use direct quotes sparingly and always indicate when you are quoting another source and cite the source. When in doubt, refer to the AS&T Language Skills Handbook for clarification.

You are to work on this test individually. While discussing the questions with others to clarify them is permissible, solutions are to be done individually.

1. Explain how both the waterfall model of the software process and the prototyping model can be accommodated in the spiral process model.

2. Describe what is meant by the terms cohesion, coupling, and adaptability as design criteria. Explain why maximizing cohesion and minimizing coupling leads to more maintainable systems.

3. Develop a class object model, using the notation described in class, for the Recycling Project Description attached.

4. The design of a spelling checker is discussed in Appendix A of Sommerville (section 4, pages 617-619) to illustrate design description languages. Document this design by producing:
   (a) a context diagram;
   (b) a leveled and balanced set of data flow diagrams;
   (c) a data dictionary; and
   (d) a structure chart.

RECYCLING PROJECT DESCRIPTION

A system is needed to control a recycling machine for returnable glass bottles, plastic bottles and metal cans. The machine can be used by up to three customers at the same time and each customer can return all three types of items. These items come in various types and sizes. The machine must check which type of item was turned in so that it can print a receipt. A receipt, which can be taken to a cashier, will be printed out. The total value of the items turned in will be printed on one line and the value of each item type will be printed on separate types for each line.

The machine has to be maintained so there is information for the maintenance operator which consists of the total quantity of each item type that has been turned in since the last time the totals were cleared. This information should be able to be printed out. In addition to these totals, the maintenance operator should be able to change the values assigned to individual item types. The machine has numerous mechanical functions which can go awry. The machine has an alarm which indicates that an item is stuck or that the receipt roll is out of paper.

To return items the customer first presses the receipt button to clear all totals. The system then places the items into the correct item type slots. With each item deposited the machine increases the daily totals and the customer totals for that item type. The customer presses the receipt button again to indicate the end of his transaction. The action prints the receipt and updates the daily totals.

The operator needs the ability to turn the alarm off, print the daily reports, and clear the report totals. Not only can the value of the items be changed, but because manufacturers regularly change their packaging, the operator must be able to change the allowable sizes for each item type. When items are stuck the customer is prevented from inserting more items but that customers totals are not lost. After the stuck item is cleared from the machine, the customer can continue to insert items which are added to his/her previous totals.

CSCI 3250-201: SOFTWARE ENGINEERING

STUDY GUIDE FOR TEST 1, PART 2 - IN-CLASS

1. Sommerville says "assuming the software provides the required functionality, there are four key attributes which a well-engineered software system should possess". What are they?

2. Name and briefly describe/compare the three types of software maintenance activities.

3. Familiarize yourself with the requirements of the attached <u>Client Request: Small College Registration System</u>.

4. What are the fundamental differences between object-oriented and function-oriented design?

5. Regarding verification and validation:
   (a) What is the difference between verification and validation and why is validation a particularly difficult process?
   (b) What is the difference between testing and debugging?
   (c) Define the following types of testing: unit, module, sub-system, system, acceptance, alpha, beta, regression.
   (d) Understand these testing strategies: top-down, bottom-up, thread, stress.
   (e) Are V&V and Software Quality Assurance synonyms? Explain.

6. Understand these object-oriented terms: class, object (state, behavior, identity), abstraction, encapsulation, modularity, hierarchy (2 types), inheritance, polymorphism.

7. Transform analysis is a strategy for obtaining a first-cut structure chart from a data flow diagram. One step in transform analysis is identifying the <u>central transform</u>. What is the central transform and how can it be identified?

8. Regarding configuration management (CM):
   (a) What is CM and what are the four major CM activities?
   (b) What happens when something is placed under configuration control (i.e. becomes a configuration item)?
   (c) What are the configuration items in a well-engineered software project?

9. According to Sommerville, would a good software development organization have a Software Quality Assurance (SQA) Group responsible for SQA throughout the organization or various SQA teams, one associated with each software project? Do you agree with Sommerville? Can you think of an advantage and disadvantage of each type of SQA organization?

CLIENT REQUEST: SMALL COLLEGE REGISTRATION SYSTEM

The Dean of the Faculty at a small college wants to automate their student registration system. Following is her request.

We want to retain many of the features of our current manual registration system while automating some of the paperwork. Specifically, we want to retain the interaction between faculty and students during the registration process. We visualize the following process.

Each faculty member will continue to participate in registration. Each will sit at a table in the student center with a set of "slot cards" for each course they teach. They'll have one slot card for each position in a class (i.e. if the maximum enrollment for a class is 12, then there will be 12 slot cards for that class). A slot card contains the course number, location, time, and space for student name and instructor's signature. Each faculty member also has a blank class roster for each course containing course number, location, time, and spaces for names of students in the class.

A schedule booklet is sent to each student approximately two weeks prior to registration. At the same time each student is sent an up-to-date transcript.

On registration day, students go to the student center and pick up a slot card for each course they want to take. Before giving them a slot card, the faculty member looks at their transcript (to check prerequisites, etc) and takes one of the following actions:
(a) completes a slot card for the course (adds student name and faculty signature), adds the students name to the class roster, and gives the slot card to the student; or
(b) tells the student they can't take the course.

When finished, students turn in their slot cards and instructors turn in their class rosters. The (new) automated system is then used to officially register students. The system generates a schedule and bill for each student, an official class roster for each instructor, and a "discrepancy report" for each student or instructor, as appropriate. Discrepancies occur in two ways: first, whenever a slot card exists for a given student but that student does not appear on the class roster submitted by the instructor; and second, whenever a student does appear on the class roster submitted by an instructor but a corresponding slot card is not submitted.

CSCI-3250, TEST 1 PART II NAME: ______________________

1. (5 points) (a) Describe the four key attributes of well-engineered software according to Sommerville.

(b) In addition, a well-engineered software system must ________
____.

2. (6 points) A software organization has successfully developed and marketed a word-processing package. Though this may seem impossible, imagine that they have done such a thorough job that the specifications have been completely met and tested and there are absolutely zero defects in the released software. Under these conditions, will any maintenance activity be expected? If so, what type? Explain thoroughly and use examples to illustrate your points.

3. (5 points) (a) Define coupling. What is the major design goal for coupling?

(b) Define cohesion. What is the major design goal for cohesion?

(c) What is relationship, if any, between coupling and cohesion; i.e. do they have any effect on one another?

4. (8 points) There are two types of hierarchical relationships between classes; Generalization-Specialization ("is a") and Whole-Part ("has a").

(a) Using Rumbaugh's notation (as on the take-home test), show an example of each type of hierarchy.

(b) What is the key difference between these two types of hierarchical relationships?

5. (9 points) (a) Distinguish between verification and validation (V&V). Illustrate by giving an example of a verification activity and an example of a validation activity.

(b) Distinguish between V&V and software quality assurance.

(c) Distinguish between testing and debugging.

(d) Consider this statement: One of the goals of software testing is to prove that a program works correctly.
(1) Is the statement true or false?

(2) If true, what are the <u>other</u> goals of software testing? If false, what <u>are</u> the goals of software testing.

6. (17 points) For each item, write the letter of the correct answer in the space provided.

___ A. The modules of a well designed system will have a ______ level of coupling and a ______ level of cohesion.
(a) high, high (c) low, high (e) none of these
(b) high, low (d) low, low

___ B. Stamp and control coupling should always be avoided.
(a) True (b) False

___ C. Factors in the level of coupling between modules include
(a) amount of information passed (d) a & b, not c
(b) complexity of interface (e) a & c, not b
(c) amount of control of one module over another (f) b & c, not a
(g) a & b & c

___ D. The most desirable form of coupling is
(a) control (b) content (c) data (d) stamp (e) common

___ E. The most desirable form of cohesion is
(a) coincidental (d) communicational (g) temporal
(b) logical (e) functional
(c) procedural (f) sequential

___ F. If a module performs several different, but related, activities then its cohesion is not ideal. The different activities could be related due to "the data of the problem", "due to time", or "due to falling in some general category." Which is most preferable?
(a) related by general category (c) related by time
(b) related by data (d) all equally bad

___ G. By itself, a structure chart can give one a pretty good idea of the coupling and cohesion of the system.
(a) True (b) False

___ H. DFD's are __________ documents; structure charts are ____________documents.
(a) analysis, analysis (c) analysis, design (e) none of these
(b) design, design (d) design, analysis

___ I. Which is not shown by a structure chart?
(a) modules within system (d) module hierarchy
(b) interfaces between modules (e) function of modules
(c) execution order of modules (f) all are shown

___ J. The detection of errors associated with the interfaces between modules is a goal of which type of testing?

(a) module (c) validation (d) system
(b) integration (d) regression

__ K. A final set of tests to ensure that the software meets all requirements is called ____________ testing.
(a) module (c) validation (d) system
(b) integration (d) regression

__ L. Errors are often introduced while performing perfective maintenance. The type of testing designed to detect such errors is
(a) module (c) validation (d) system
(b) integration (d) regression

__ M. Advantages of top-down testing include
(a) allows early verification of overall control logic
(b) makes providing of test cases easier
(c) allows early verification of major module interfaces
(d) a and b only (f) b and c only (h) none of
(e) a and c only (g) a, b, and c the above

__ N. ____________ is the ability of two or more classes to respond to the same message, but each in its own way.

(a) abstraction	(e) hierarchy	(i) polymorphism
(b) behavior	(f) identity	(j) state
(c) class	(g) inheritance	(k) none of these
(d) encapsulation	(h) object	

__ O. ____________ is a set of objects with a common structure and a common behavior.

(a) abstraction	(e) hierarchy	(i) polymorphism
(b) behavior	(f) identity	(j) state
(c) class	(g) inheritance	(k) none of these
(d) encapsulation	(h) object	

__ P. ____________ is ignoring certain details of a subject that are not relevant to the current purpose in order to focus on other aspects that are.

(a) abstraction	(e) hierarchy	(i) polymorphism
(b) behavior	(f) identity	(j) state
(c) class	(g) inheritance	(k) none of these
(d) encapsulation	(h) object	

__ Q. ____________ is the act of grouping into one object both data and the operations that affect that data.

(a) abstraction	(e) hierarchy	(i) polymorphism
(b) behavior	(f) identity	(j) state
(c) class	(g) inheritance	(k) none of these
(d) encapsulation	(h) object	

CSCI-3250 SOFTWARE ENGINEERING
TEST 2 STUDY GUIDE

The following is a guide to review material covered in class and/or in assigned reading. You are <u>not</u> to turn in answers but rather use the guide in preparing for the 4/20/93 test.

1. Sommerville says that a major difficulty is establishing large software system requirements is that the problems are usually "wicked problems."

   a) What does he mean by the term wicked problem?

   b) What are some examples?

2. Distinguish between a requirements definition and a requirements specification? Why is it useful to draw a distinction, as Sommerville does, between a requirements definition and a requirements specification?

3. Distinguish between functional requirements and non-functional requirements?

4. Distinguish between a logical (or essential) model and a physical model of a system?

5. Regarding a Software Project Management Plan (SPMP):

   a) What sorts of things go into a SPMP?

   b) Is it a managerial document or a technical document?

   c) What is the relationship between the deliverables of your teams (on the class project) and a SPMP?

6. Consider examples such as that described on page 58 (and Figures 3.6, 3.7, in writing/improving requirement definitions.

7. Consider Exercises 5.2 and 5.8 (as discussed in class).

< OVER >

8. Regarding software reviews, or walkthroughs:

    a) What is the purpose?

    b) Who participates?

    c) What are the different roles?

    d) What are some standard guidelines?

9. Regarding software reliability:

    a) What is it?

    b) What are some of the problems with specifying reliability; why is it difficult?

    c) What is the relationship, if any, between <u>process</u> and <u>product</u> reliability?

    d) What metrics exist for assessing software reliability?

10. Describe the purpose and the steps of <u>statistical testing</u>.

11. Consider Exercise 20.4 (as discussed in class).

12. Everyone should be <u>generally</u> familiar with the <u>purpose, responsibilities, and deliverables</u> of each of these teams on the class project:

    a) configuration management;

    b) requirements;

    c) users manual;

    d) test plan;

    e) preliminary design.

    In addition everyone should have <u>detailed</u> knowledge of the <u>purpose, responsibilities, and deliverables</u> of the team(s) of which you are a member.

CSCI-3250, TEST 2 NAME:

1. (15 points) Concerning walkthroughs or reviews:

   (a) React to this statement: The purpose of a review/walkthrough is to review a work product in order to identify and correct errors.

   (b) Who are the participants?

   (c) When is it appropriate, during a project, to hold a structured walkthrough?

   (d) Assume you're the leader of a project team and you're preparing a "Guide to Structured Walkthroughs" for your project team. List 5 key guidelines that you would include on preparing and conducting a structured walkthrough.

      1.

      2.

      3.

      4.

      5.

2. (10 points) (a) Is your project in this class (the Computer and Information Sciences Student Records System) a "wicked problem" according to Sommerville's definition? Answer yes or no and then justify your answer.

   (b) Give an example of a wicked problem. (NOTE: the example cannot be one discussed in the textbook.)

3. (10 points) (a) Distinguish between requirements and specifications.

   (b) Consider the requirements phase and the specification phase of a software development project. Would it make more sense to combine these activities into one phase, rather than treating them as two separate phases? Justify your answer.

4. (10 points) (a) What is the difference between functional requirements and non-functional requirements?

   (b) Give an example of a functional requirement and of a non-functional requirement from the class project (the Computer and Information Sciences Student Records System).

5. (12 points) Rewrite each requirements so that it may be objectively validated. Make any reasonable assumptions.

   (a) An on-line "HELP" function will provide users of the Computer and Information Sciences Student Records System with appropriate help.

   (b) The HVAC monitoring system must significantly reduce electrical consumption throughout the organization.

   (c) The network must be set up at a reasonable cost.

   (d) The system should be easy for experienced controllers to use and should be organized so that user errors are minimized.

6. (6 points) Briefly explain the difference between a logical and a physical model of a system.

7. (14 points) Answer each of the following True or False.

___ A Software Project Management Plan (SPMP) is a managerial document rather than a technical document.

___ A SPMP for the class project (Computer and Information Sciences Student Records System) would include a description of the team organization and the responsibilities of each team.

___ A SPMP for the class project (Computer and Information Sciences Student Records System) would include a description of the design strategy to be used (for example, structured design or object-oriented design).

___ A SPMP for the class project (Computer and Information Sciences Student Records System) would include a description of the configuration management procedures.

___ A good software process cannot guarantee cannot guarantee a reliable software product.

___ According to Sommerville, a good software process is essential to the development of reliable software.

___ The purpose of statistical testing is to estimate software reliability.

8. (15 points) Regarding software reliability:
   (a) Define it.

   (b) Distinguish between process reliability and product reliability.

   (c) Name and briefly describe four metrics used to assess software reliability.

      1.

      2.

      3.

      4.

   (d) For each of the metrics described in part (c), to what type of software system is it most relevant?

CSCI-3250 SOFTWARE ENGINEERING
STUDY GUIDE FOR TEST 3

1. Review material from both parts of Test 1 (take-home part and in-class part) and the study-guide for Test 1.

2. Review material from Test 2 and the study guide for Test 2.

3. Questions regarding your second project, the Computer and Information Sciences Student Records System, will be included.

4. **

   The following question will definitely be on the test. Prepare your answer in advance, using a word processor, and submit it when the test is given. It will count 15 points on the test.

   ****

   Write a client request for a project that you think would be suitable to serve as the first project for a future offering of this course (e.g. comparable to the EMS-911 and the Recycling System projects used in this class). The client request must be between one-half page and one-full page (single spaced) in length. Your solution will be graded on writing, appropriateness of the project, and originality.

The following are review questions for material covered since the second test.

---

5. Regarding software safety: a) Define these terms:

software safety	mishap	hazard
damage	risk	hazard analysis

   primary safety-critical software

   secondary safety-critical software

   b) Distinguish between software safety and software reliability?

   c) Could a software system be reliable but not safe? Explain and give an example. Could a software system be safe but not reliable? Explain and give an example.

d) One method of analyzing safety requirements is through Fault-Tree-Analysis (FTA). What is the purpose of FTA and what steps are involved?

6. Regarding ethical issues:

a) What are the defining characteristics of a software professional?

b) Define professional computer ethics.

c) Describe one case, not discussed in class, which involves an issue in computer ethics and state what you think the ethical issue is.

CSCI-3250, TEST 3 NAME:

1. (7 points) (a) Define configuration management.

   (b) What does it mean to make something a configuration item?

   (c) Name four configuration items from CISSRS.

2. (10 points) (a) Name and describe the three categories of software maintenance. In your description, make sure that the distinction between them is made clear.

   (b) Which type of maintenance would you expect to be most prevalent in the first six months following the release of CISSRS? Explain.

(c) Which type of maintenance would you expect to be most prevalent after CISSRS has been in operation for a year? Explain.

3. (6 points) We have discussed several software life cycle models, including the waterfall model and the spiral model.

(a) What are the major differences between the waterfall model and the spiral model?

(b) For what type of system would the waterfall model be appropriate? Give an example.

(c) For what type of system would the spiral model be appropriate? Give an example.

4. (6 points) In object-oriented analysis and design, there are two types of hierarchical class relationships; Whole-Part ("has a") and Generalization-Specialization ("is a").

(a) What is the major difference between the two?

(b) Using Rumbaugh's notation, show the relationship between students, advisor-mentors, clerical staff, and super advisor-mentors in CISSRS.

5. (16 points) Briefly distinguish between:
   (a) preliminary design and detailed design (relative to CISSRS)

   (b) software safety and software reliability

   (c) abstraction and encapsulation

   (d) acceptance testing and beta testing

6. (4 points) Classify the following as a Verification activity, a Validation activity, or neither.

   ____________ prototyping

   ____________ defect testing

   ____________ the Users Manual Review for CISSRS

   ____________ the Test Plan Review for CISSRS

7. (7 points) (a) You've been asked to prepare <u>specifications</u> for structured walkthroughs. Write 4 <u>specifications</u> that you would include.

   (b) A standard guideline is that identified problems are <u>not</u> to be corrected during the walkthrough. Why?

8. (4 points) (a) What is a hazard?

   (b) What is the output of hazard analysis?

   (c) When is hazard analysis undertaken?

9. (5 points) (a) What are the defining characteristics of a software professional?

   (b) Define professional computer ethics.

   (c) Describe a case, not discussed in class, which involves an issue in computer ethics and state what you think the ethical issue is.

10. (10 points) The following describes how a small college bookstore orders (buys) and sells textbooks.

On a separate sheet of paper:

(a) Construct the context diagram;
(b) Construct the first level DFD;
(c) Circle the central transform.

The Bookstore maintains an inventory card for each course in the catalog. An inventory card contains the title, author, and publisher of the textbook currently used as well as the number currently in stock.

In April the Bookstore asks each department to provide them with the following information for each course they teach:

(a) title, author, publisher of book to be used next year;
(b) expected enrollment in the course next year.

In June, the Bookstore creates a <u>Books Needed File</u> and a <u>Buy Back File</u>. The Books Needed File contains the title, author, publisher, and "number needed" for each book. The "number needed" is expected enrollment minus the number in stock. The Buy Back File contains title, author, publisher, and number needed for each book for which the expected enrollment exceeds the number in stock.

During June the Bookstore will buy books from students as long as a book is in the Buy Back File. Of course each time a book is bought from a student the number of copies (of that particular book) is reduced by one and the book is removed from the Buy Back File when/if the number needed becomes zero.

In July the Bookstore prepares an Order List containing the title, author, publisher, and number to be ordered for each book that is still needed. The Order List is then used to create an individual Book Order Form for each publisher. These Book Order Forms are sent to the publishers.

SECOND SEMESTER EXAMINATION

REVIEW SHEET FOR TEST

Ada
- classes of subsystems resulting from the analysis phase
- as a design tool, support for reuse
- language features and constructs
  - packages, package specifications
  - named association
  - overloading
  - compilation units
  - context clauses
  - I/O, files (text, sequential, direct access)
  - program structure
  - data types
  - statements (loops, selection, null, block, etc)
  - program units (procedures & functions, packages, tasks, generics)
  - exceptions, exception handlers

Requirements
- distinction between requirements and specifications
- ways to evaluate a requirements specification (context analysis, walkthroughs, inspections, requirements validation table)
- steps in developing requirements
- validation of requirements, requirements validation table, formal specifications
- 2167a as a standard (2167a language: CSCI, CSC, HWCI, CSU, IRS)

Structured analysis and design
- transforms, data flows, leveling, balancing
- control flows
- process specifications (what? where used? methods, notations)
- transform analysis and transaction analysis (purpose, technique, input, output)

Design
- interface design
- criteria for good design
- coupling (definition, design goals, levels)
- cohesion (definition, design goals, levels)
- distinction between preliminary design and detail design (goals, methods,deliverables of each)

- object-oriented design (OOD)
  - Distinguish between functional design and OOD.
  - goals, terminology, characteristics (classes, sub-classes, objects, inheritance, information hiding, abstraction, encapsulation, methods, aggregation)
  - methods of identifying objects
  - Rumbaugh notation

Configuration management (CM)
- CM activities
- configuration items, baselining
- implementation of CM, change control boards

Verification and validation (V&V)
- objectives
- distinction between verification and validation
- verification - static and dynamic
- validation - static and dynamic
- V&V activities at each development phase
- testing
  - relationship to V&V
  - levels of testing (unit, integration, acceptance)
  - black-box, white-box methods
  - test plans
  - test oracle, test harness
  - reliability - faults and failure

Software Quality Assurance (SQA)
- definition; SQA activities
- relationship between SQA, V&V, CM
- software quality: what is it? components of
- reviews (walkthroughs, inspections)
  - purpose
  - what is reviewed
  - participants, roles
  - guidelines, standards

Project 2
- Responsibilities, functions, roles, and deliverables of all teams.
- Application of lecture and reading material to project.

Test

1. (14 points) For each definition, write the letter corresponding to the term (from the list below) which the statement describes. A term cannot be used more than once.

___ 1. organizational unit within company that reviews each request for a change
___ 2. type of analysis technique used in V&V which is the manual or automated examination of the product
___ 3. process of evaluating software at the end of the software development process to ensure compliance with software requirements
___ 4. strategy for developing test cases where the test cases focus on requirements (functionality, input, output)
___ 5. set of objects with a common structure and behavior
___ 6. relationship among classes by which one class shares the structure or behavior defined in another class
___ 7. principle that each program unit should only be allowed access to data or procedures that are required for the unit to perform its function
___ 8. the only way objects communicate and request services from other objects
___ 9. a measure of the dependency between components
___10. first level of testing; involving individual components
___11. testing designed to push a system beyond its limits in order to observe the system's failure behavior
___12. a graphical notation for representing algorithms in detailed design
___13. predicted results for a set of test cases
___14. a strategy (to create a structure chart from data flow diagrams) based on the idea that most systems have a transform center

TERMS FOR MATCHING QUESTION 1

A. black box testing	O. object
B. Change Control Board (CCB) testing	P. static analysis
C. class	Q. stress testing
D. cohesion	R. system testing
E. coupling	S. test oracle
F. dynamic analysis	T. thread testing
G. exhaustive testing	U. transaction analysis
H. formal analysis	V. transform analysis
I. information hiding	W. unit testing
J. inheritance	X. validation
K. integration testing	Y. verification
L. message	Z. white-box testing
M. method	
N. Nassi-Shneiderman chart	

2. (16 points) Matching. The answers at the right may be used more than once.

___ 1. only programming unit in which private data types may be declared
___ 2. programming unit which is used as the main program or driver of a software system
___ 3. programming unit that operates in parallel with other programming units
___ 4. data type which provides a new name for another (potentially constrained) data type
___ 5. data type which defines a distinct data type and inherits all properties of its base type
___ 6. processing of declarations at execution time
___ 7. type of array where lower and upper bounds are known at type declaration time
___ 8. when an entity in Ada has more than one meaning
___ 9. the process of detecting an exception and alerting calling subprogram
___10. predefined Ada package which provides input-output of values of any nonlimited type in a binary format where elements are read and written using an index
___11. predefined Ada package which provides input-output for enumeration types
___12. the ability to compile a program in pieces with full compiler checking
___13. predefined Ada package which provides input-output of values in human-readable form
___14. only programming unit whose body is optional
___15. control construct that promotes flexible response to run-time events such as hardware overflow
___16. new program unit tailored to a particular application of a generic

A. constrained array
B. context clause
C. derived type
D. **Direct_IO**
E. elaboration
F. exception
G. exception handler
H. function
I. instantiation
J. overloading
K. package
L. procedure
M. raising exception
N. separate compilation
O. **Sequential_IO**
P. subtype
Q. task
R. **Text_IO**
S. unconstrained array

3. (13 points) Short answer - Ada.

A. (3 points) Based on the following configuration, list the order that the programming units must be compiled. Distinguish between the specification and body of the packages when listing the compilation order. If the following configuration is illegal in Ada, indicate "illegal" and specify why it is a problem.

```
package BROWN uses packages GREEN and YELLOW
package GREEN uses package RED
package YELLOW uses packages GREEN and BLUE
package RED uses package BLUE
```

B. (4 points) Indicate the possible data types of the actual parameter which matches the formal parameter for the following generic. Also list the operations and attributes which are available inside the generic body for this formal parameter.

```
generic
    type ELEMENT is (<>);
package EXAMPLE is
    -- rest of generic package
end EXAMPLE;
```

C. (2 points) How many exception handlers are executed if MAIN is called? ______

```
procedure MAIN is
        ERROR1, ERROR2, ERROR3 : exception;

        procedure B is
        X : Integer;
        begin
                        X := 1;
                        if X = 1 then
                          raise ERROR2;
                        else
                          raise ERROR3;
        exception
                        when ERROR2 => raise;
        end B;

        procedure C is
        begin
                        B;
        exception
                        when ERROR2 => raise ERROR1;
                        when ERROR1 => null;
                        when others => raise ERROR2;
        end C;

        procedure D is
        begin
                        C;
        exception
                        when ERROR3 => raise;
                        when ERROR1 => raise ERROR3;
        end D;

begin
        D;
exception
        when ERROR3 => null;
        when ERROR2 => PUT_LINE("ERROR IN PROCEDURE C");
        when ERROR1 => PUT_LINE("ERROR IN PROCEDURE B");
end MAIN;
```

D. (4 points) Discuss the tradeoffs of the two alternative specifications of ORDER_MAKER below:

```
generic
        type ITEM is private;
        with function "<" (LEFT, RIGHT : ITEM) return Boolean
                       is <>;
procedure ORDER_MAKER (I,J : in out ITEM);


generic
        type ITEM is (<>);
procedure ORDER_MAKER (I,J : in out ITEM);
```

IV. Short discussion

A. (6 points) (a) Describe the relationship between coupling and cohesion. (Do not simply define coupling and cohesion).

(b) Distinguish between data coupling, stamp coupling, and control coupling?

(c) What is the most desirable type of cohesion, and why?

B. (12 points) (a) Distinguish between preliminary design and detailed design. Your answer must include the general goals and the general deliverables of each.

(b) Name and briefly describe the preliminary design deliverables for project 2 (Third Eye).

or

(b) Name and briefly describe the detailed design deliverables for project 2 (Third Eye).

(c) Describe verification and validation techniques used during design.

C. (15 points) (a) Explain the relationship between testing, software quality assurance (SQA), and V&V, including the distinction between them.

(b) Distinguish between white-box and black-box testing, including the purpose of each and the relationship between them.

(c) Briefly describe two white-box methods and two black-box methods.

D. (5 points) (a) React to this statement: The purpose of a review/walkthrough is to review a work product in order to identify and correct errors.

(b) Who are the participants in a review/walkthrough and what are their roles?

E. (5 points) Describe (a) the purpose of 2167a, (b) its relationship to requirements and specifications, and (c) a description of three of its major components.

F. (3 points) Context analysis is one technique used to elicit complete and accurate requirements. Describe the process of context analysis and explain what it adds to the process of abstracting requirements from the customer request into a requirements list.

G. (4 points) Name and describe two ways of identifying objects that were discussed in class.

Test KEY

1. (14 points) For each definition, write the letter corresponding to the term (from the list below) which the statement describes. A term cannot be used more than once.

_B_ 1. organizational unit within company that reviews each request for a change
_P_ 2. type of analysis technique used in V&V which is the manual or automated examination of the product
_X_ 3. process of evaluating software at the end of the software development process to ensure compliance with software requirements
A,F 4. strategy for developing test cases where the test cases focus on requirements (functionality, input, output)
_C_ 5. set of objects with a common structure and behavior
_J_ 6. relationship among classes by which one class shares the structure or behavior defined in another class
_I_ 7. principle that each program unit should only be allowed access to data or procedures that are required for the unit to perform its function
_L_ 8. the only way objects communicate and request services from other objects
_E_ 9. a measure of the dependency between components
_W_10. first level of testing; involving individual components
_Q_11. testing designed to push a system beyond its limits in order to observe the system's failure behavior
_N_12. a graphical notation for representing algorithms in detailed design
_S_13. predicted results for a set of test cases
_V_14. a strategy (to create a structure chart from data flow diagrams) based on the idea that most systems have a transform center

TERMS FOR MATCHING QUESTION 1

A. black box testing	O. object
B. Change Control Board (CCB) testing	P. static analysis
C. class	Q. stress testing
D. cohesion	R. system testing
E. coupling	S. test oracle
F. dynamic analysis	T. thread testing
G. exhaustive testing	U. transaction analysis
H. formal analysis	V. transform analysis
I. information hiding	W. unit testing
J. inheritance	X. validation
K. integration testing	Y. verification
L. message	Z. white-box testing
M. method	
N. Nassi-Shneiderman chart	

2. (16 points) Matching. The answers at the right may be used more than once.

_K_ 1. only programming unit in which private data types may be declared
_L_ 2. programming unit which is used as the main program or driver of a software system
_Q_ 3. programming unit that operates in parallel with other programming units
_P_ 4. data type which provides a new name for another (potentially constrained) data type
_C_ 5. data type which defines a distinct data type and inherits all properties of its base type
_E_ 6. processing of declarations at execution time
_A_ 7. type of array where lower and upper bounds are known at type declaration time
_J_ 8. when an entity in Ada has more than one meaning
_M_ 9. the process of detecting an exception and alerting calling subprogram
D 10. predefined Ada package which provides input-output of values of any nonlimited type in a binary format where elements are read and written using an index
R 11. predefined Ada package which provides input-output for enumeration types
N 12. the ability to compile a program in pieces with full compiler checking
R 13. predefined Ada package which provides input-output of values in human-readable form
K 14. only programming unit whose body is optional
F 15. control construct that promotes flexible response to run-time events such as hardware overflow
I 16. new program unit tailored to a particular application of a generic

A. constrained array
B. context clause
C. derived type
D. **Direct_IO**
E. elaboration
F. exception
G. exception handler
H. function
I. instantiation
J. overloading
K. package
L. procedure
M. raising exception
N. separate compilation
O. **Sequential_IO**
P. subtype
Q. task
R. **Text_IO**
S. unconstrained array

3. (13 points) Short answer - Ada.

A. (3 points) Based on the following configuration, list the orde[illegible] the programming units must be compiled. Distinguish between the specification and body of the packages when listing the compilation order. If the following configuration is illegal in Ada, indicate "illegal" and specify why it is a problem.

package BROWN uses packages GREEN and YELLOW
package GREEN uses package RED
package YELLOW uses packages GREEN and BLUE
package RED uses package BLUE

1. Blue spec
2. Red spec
3. Green spec
4. Yellow spec
5. Brown spec
6. bodies in any order

$$\begin{tikzcd}
\textbf{Red} \arrow[r] & \textbf{Blue} \\
\textbf{Green} \arrow[u] & \textbf{Yellow} \arrow[u] \arrow[l] \\
\textbf{Brown} \arrow[u] \arrow[ur] &
\end{tikzcd}$$

B. (4 points) Indicate the possible data types of the actual parameter which matches the formal parameter for the following generic. Also list the operations and attributes which are available inside the generic body for this formal parameter.

```
generic
    type ELEMENT is (<>);
package EXAMPLE is
    -- rest of generic package
end EXAMPLE;
```

Discrete data type

Attributes 'First, 'Last, 'Succ, 'Pred, 'Range

Operations :=, =, /=

C. (2 points) How many exception handlers are executed if MAIN is called? __4__

```
procedure MAIN is
        ERROR1, ERROR2, ERROR3 : exception;

        procedure B is
        X : Integer;
        begin
                        X := 1;
                        if X = 1 then
                          raise ERROR2;
                        else
                          raise ERROR3;
        exception
                        when ERROR2 => raise; *** 1 ***
        end B;

        procedure C is
        begin
                        B;
        exception                        *** 2 ***
                        when ERROR2 => raise ERROR1;
                        when ERROR1 => null;
                        when others => raise ERROR2;
        end C;

        procedure D is
        begin
                        C;
        exception                        *** 3 ***
                        when ERROR3 => raise;
                        when ERROR1 => raise ERROR3;
        end D;

begin
        D;
exception                                *** 4 ***
        when ERROR3 => null;
        when ERROR2 => PUT_LINE("ERROR IN PROCEDURE C");
        when ERROR1 => PUT_LINE("ERROR IN PROCEDURE B");
end MAIN;
```

D. (4 points) Discuss the tradeoffs of the two alternative specifications of ORDER_MAKER below:

```
generic
        type ITEM is private;
        with function "<" (LEFT, RIGHT : ITEM) return Boolean
                        is <>;
procedure ORDER_MAKER (I,J : in out ITEM);

generic
        type ITEM is (<>);
procedure ORDER_MAKER (I,J : in out ITEM);
```

As a generic (template):

First is broader, more flexible:
(a) allows any data type, including user defined, in instantiation
(b) provides "<"

Second is more limited in that it only works on discrete types (integer, enumeration)

IV. Short discussion

A. (6 points) (a) Describe the relationship between coupling and cohesion. (Do not simply define coupling and cohesion).

Cohesion is measure of internal strength of a component; determined by how well the parts of a component contribute to single purpose. Goal is to maximize cohesion. Coupling is a measure of dependency between 2 components; of strength of interconnections. Goal is to minimize coupling.

Strong cohesion, loose coupling, and are complementary criteria; go hand-in-hand. Increasing cohesion decreases coupling & vice-versa. Strongly cohesive components are highly independent; loosely coupled components have minimal dependency. Thus by maximizing cohesion and minimizing coupling, we reduce the number of components impacted by a change or defect; thereby simplifying maintenance. Stated another way, we reduce the chance that a change to one component will require a change to others; and we reduce "side-affects."

(b) Distinguish between data coupling, stamp *coupling, and control coupling?*

Data coupling - most desirable; only parameters communicated

Stamp coupling - communicates at least one data structure (and the called component operates on some but not all of the data structure's components

Control coupling - communicate at least one element of control

(c) What is the most desirable type of cohesion, and why?

Functional - module performs single task and each part of the module is necessary to perform that task.

Desirable because such modules are easier to maintain (less chance of change impacting other modules), and better chance of reuse.

B. (12 points) (a) Distinguish between preliminary design and detailed design. Your answer must include the general goals and the general deliverables of each.

Preliminary	Detailed
Move from problem space of requirements to solution space	Expand solution space details sufficiently to be easily implementable on computer
general SW architecture to meet specs	expand on architecture
components & their interfaces	algorithms for components
data structures	data structures refined
user interface [Mynatt]	
OOD: class design objects and interfaces object relations	OOD: internal object structure data structure dictionary algorithms for methods

(b) Name and briefly describe the preliminary design deliverables for project.

Preliminary	Detailed
Rumbaugh object model objects, associations attributes, operations	Nassi-Shneidermann charts
Object dictionary Object name, attributes, Method description, other objects	Traceability matrix ds to object attribute NS to object operations
Object requirements traceability matrix	Data structure dictionary Object class, Ada package, ds name - attr covered ds description (a la DD)
Ada specifications	
User interface (menus, reports)	

(c) Describe verification and validation techniques used during design.

Reviews: design documents (PDR, DDR)
project V&V plan
test plans

Testing: generation of test plans for unit, design unit, and system testing; generation of design-based test cases; generation of test plan for acceptance testing

Traceability of design (to requirements; to acceptance)

1. Detailed design	1. Use Nassi-Sh to trace PD
2. Use object model to trace scenarios	2. All objects in PD used?
3. Check matrix correctness/completeness; all objects related to at least one other	3. Trace matrix
4. Verify with end user	4. No interface modification

C. (15 points) (a) Explain the relationship between testing, software quality assurance (SQA), and V&V, including the distinction between them.
All are concerned with quality.
SQA encompasses V&V; V&V encompasses testing.

Verification - does system meets specifications (build product right?); validation - does implemented system meet customer expectations (build right product?).

(b) Distinguish between white-box and black-box testing, including the purpose of each and the relationship between them.
White-box (structural)
tests derived using knowledge of program's implementation; focuses on the logic of the module to determine what tests to run.
Black-box (functional)
tests derived from program's specification; designed to test the functions of the module. For a given input, does it produce the expected output.

Are not alternative approaches, but are complementary, designed to uncover different types of defects. Both are necessary; neither white-box nor black-box is sufficient by itself. White-box testing examines the existing code and thus cannot discover extra fuctions included in code but not included in requirements. Black-box can't be complete.

(c) Briefly describe two white-box methods and two black-box methods.

White-box (structural)
- Statement coverage - every statement executed at least once
- Decision coverage - every branch traversed at least once
- Condition coverage - each condition in a decision takes on all possible values at least once
- Path coverage - every path traversed at least once
Black-box (functional)
Equivalence partitioning - involves determining which input classes have common properties (are similar on some relevant dimension); then dividing inputs and outputs into equivalence partitions; and testing one item from each partition.
Cause-effect strategy - tests combinations of inputs; causes (inputs) and effects (outputs) are identified.
Boundary values strategy - involves identifying and testing conditions directly on, above, and below the edges of input equivalence classes.

D. (5 points) (a) React to this statement: The purpose of a review/walkthrough is to review a work product in order to identify and correct errors.

Statement inaccurate; purpose is to identify but not to correct errors. Reviews are QA activities; intended to identify problems; to collectively review a work product to assure that it meets requirements and standards.

(b) Who are the participants in a review/walkthrough and what are their roles?

Review leader/ moderator:	**evaluate items for readiness** **distribute materials in advance** **review material prior to meeting** **schedule review, prepare agenda**
Recorder:	**record important issues raised in review**
Producer:	**walks through product**
Reviewers reader:	**review material prior to meeting; prepare list of items misunderstood and items incorrect**

E. (5 points) Describe (a) the purpose of 2167a, (b) its relationship to requirements and specifications, and (c) a description of three of its major components.

(a) Purpose - 2167a is DoD's software development process standard. Treats software development as a milestone driven project. Milestones are generally documents or clearly specified events. 2167a characterizes several processes separated by the completion of milestones.

(b) SRS - distinguish between requirements & specifications
- requirements; classify (mandatory, desirable, unessential, stability)
- specifications - default conditions, handle errors

(c) SRS - software requirements specification
IRS - interface requirements specification
CSCI - computer software CI
HWCI - hardware CI
CSC - computer software component
CSU - computer software unit

Systems contain segments, segments contain configuration items, configuration items contain configuration components and components contain units.

F. (3 points) Context analysis is one technique used to elicit complete and accurate requirements. Describe the process of context analysis and explain what it adds to the process of abstracting requirements from the customer request into a requirements list.

a) **Interview customer/user to understand domain; questions such as:**
   Why develop or want this system?
   Where will it be used?
   Who will use it?
   What are the economic and operational boundary conditions of its use?
   This results in software needs.

b) **Adds items to requirements list which were not explicitly stated in the abstract; i.e. it reveals constraints and unstated requirements.**

G. (4 points) Name and describe two ways of identifying objects that were discussed in class.

Linguistic analysis: **nouns are potential objects and/or attributes.**

verbs are potential opreations

abverbs are potential constraints on operators

Use cases (scenarios; sequence of events in interacting with system)

Requirements documents (CD, DFDs, DD)

VII Ada

I INTRODUCTION

We used Ada as an entire life-cycle tool throughout the two-semesters. Our approach of a coordinated blending of lectures and projects and the use of three techniques in teaching Ada - a spiral approach, program reading, and a detailed examination of language features - enabled students to see Ada's support for software engineering in and beyond implementation.

Our approach uses three distinct teaching techniques: a spiral approach, program reading, and a detailed examination of language features. The spiral approach to teaching involves the gradual introduction and application of concepts and the revisiting of them in increasing depth throughout the remainder of the course in both lectures and project activities. Applying this spiral approach, Ada is introduced as software engineering concepts (e.g., abstraction, information hiding, reusability, and maintainability) are introduced. With each concept, the language features of Ada which support that concept are also introduced. For example, during the discussion of maintainability students are shown language features such as named association of parameters, overloading, packages, and separation of interface specifications. These language features are presented initially through program reading utilizing small examples and are discussed only in relation to their support of maintainability. Only after students have experienced Ada as a software development tool do they examine in detail the language features of Ada. Ada's role in the course projects is described in the following section.

II THE PROJECTS

The three projects provide the focus for the two semesters and, collectively, provide students with a variety of experiences and roles. The first project, referred to as the small project, begins in week two and extends through week seven of the first semester. The second project, referred to as the extended project, is introduced in week six of the first semester and extends through week eleven of the second semester. Finally, the third project, referred to as the maintenance project, occupies weeks six through twelve of the second semester. Note that the projects overlap and all are completed prior to the end of the second semester.

In order to provide both varied and realistic experiences, the three projects differ in a number of significant ways: size and complexity, project team organization, information provided to the teams at the start of the project, deliverables produced by the student teams, development methodology, tools, controlling disciplines, and the use of reviews. A brief description of the three projects follows.

The small project provides an early and quick immersion into the software development process. Ada is introduced, not as an implementation language, but as a specification tool used in high level design. During this project students used Ada-TUTR [1] for an initial familiarization with Ada. Ada-TUTR provides interactive instruction in the Ada programming language,- allowing the student to learn at his own pace. On a PC, it requires a hard-disk or a 3.5-inch disk. Access to an Ada compiler is helpful, but not necessary. This was the students first look at Ada. We did not discuss the details of Ada programming style until the extended project.

During the last week of the small project, the students are introduced, via a customer request, to the extended project in which Ada has an expanded role. The project is developed using Ada in the high-level design and as the implementation language. This project serves as a vehicle both to revisit concepts in depth that were briefly introduced in the small project and to introduce and utilize new concepts including detailed design, configuration management, and verification and validation. We adopted the Software Productivity Consortium's, Ada Quality and Style: Guideline for Professional Programmers, as the programming standard for the course. In our labs we use Meridian Ada for the PC.

During the first semester, the extended project is taken through the preliminary design review. Deliverables, specified in Ada, become baseline documents for detailed design beginning in the second semester. Tools which we have developed to facilitate Ada implementation (e.g., modified Nassi-Shneiderman diagrams and object traceability matrix) are introduced during the extended project. These are discussed in detail in lecture and lab materials.

The maintenance project overlaps the last five weeks of the extended project and continues one week beyond it. A large Ada artifact is provided and students must implement the change, including modification of all appropriate documents. Previous semester's extended projects could be used for the maintenance exercise. We have also used Software Maintenance Exercises for a Software Engineering Project Course, by Engle, Ford, and Korson. The maintenance project constitutes yet another circuit in the spiral from which to revisit and reinforce significant software engineering concepts, but this time from the maintenance perspective.

Thus Ada is integrated into the project experiences as well as into the lectures. Early in the first semester it is introduced through program reading techniques [2], and program reading continues throughout (see Section IV). Other activities include the writing of Ada high-level design specifications, the implementation of a system in Ada, and maintenance on an existing Ada system. This approach to teaching and learning Ada enables students to see Ada as more than an implementation language.

III THE SPIRAL APPROACH

Placing a small project at the beginning of the course permits us to start the spiral approach immediately. Lectures include a rapid overview of the elements of a software development life cycle. We emphasize design and introduce Ada as a specification language through program reading. These concepts are also emphasized in project deliverables through Ada specifications and a preliminary design review.

An extended project provides an opportunity for another pass through the lecture-project spiral. During this phase of the course all of the concepts from the first pass through the spiral are revisited and expanded upon in both lecture and project deliverables. While Ada was used only in the high-level specification of the small project, it is now used as a requirements specification and design tool as well as an implementation language.

Object-oriented design is used in the extended project. The required preliminary design deliverables include:

1. An object diagram using Rumbaugh notation,
2. A class dictionary,
3. An object-requirements traceability matrix,
4. Ada specifications for each object class, and
5. Descriptions of all major user interfaces.

Traceability is extended into detailed design by means of a detailed design traceability matrix (see Lecture 31 and associated handouts). We created this matrix to provide traceability between preliminary design, detailed design and implementation. The detailed design matrix first provides traceability between an object's attributes and its data structures and between those data structures and their Ada package representation. The matrix also provides traceability between the object's operations, the detailed design model of those operations and the Ada package embodying those operations.

We used Nassi-Shneiderman diagrams as the detailed design model for these operations. Nassi-Shneiderman diagrams provide a code independent notation for the development of algorithms which allows the students to think through the solution without considering language details. We created Nassi-Shneiderman extensions to include language features of Ada which are unique and which have no associated notation.

During the extended project, program reading is continued as examples and classroom exercises provided go into greater depth. Carefully organized use of self-paced Ada tutorial materials, laboratory experiences, and the Ada Language Reference Manual makes it possible to minimize in-class discussion of Ada syntax. In lectures, Ada's role in specification and design is emphasized. The use of accepted controlling techniques and standards introduced in the small project is also reinforced.

The maintenance project helps students see the utility of controlling techniques during original development. Ada's impact on the development of maintainable artifacts is emphasized. By equating maintenance and development, the students revisit most of the concepts previously discussed. This third trip through the spiral makes it easier for them to work with a large unfamiliar artifact. Many students find this somewhat surprising and rewarding.

IV PROGRAM READING

Program reading is an effective technique in teaching Ada because it allows the students to see software systems developed in Ada before they worry about the implementation details of the language. In program reading, students become familiar with a programming language through first reading programs rather than writing programs. Relatively large, well-designed software artifacts written in Ada are examined by the students. The students see not only the source code but also supporting documentation (e.g., analysis model, design model). Program reading emphasizes the design of the system so that the students get the "big picture" of how effective it is to build software using Ada. Language features are pointed out but discussed only in relation to effective software design. For example, as the software engineering concept of abstraction is discussed, its support in Ada is examined through program reading. Program reading is also consistent with the spiral approach since it allows the software to be examined in increasing detail as the students' knowledge increases.

V DETAILED EXAMINATION OF LANGUAGE

The detailed examination of language features is the last technique used to teach Ada. Only after students have experienced Ada as an analysis and design tool are they introduced to Ada as an implementation tool. This approach is particularly applicable to Ada because of its support throughout the life cycle while most other programming languages are merely implementation tools. Most of the language features examined at this time have already been demonstrated to the students through the program reading or examples which supported the software engineering concepts and topics discussed in the lectures. Because of this familiarity with the language, the students move very quickly through the implementation details. Because the details were covered so quickly, one of the students, Mitchell Moses, developed an Ada reference card to help his team members remember the language. He has graciously consented to allow us to include the reference card for your use.

Ada Language
Programmer's Quick Reference Card

Scalar Data Types

Type	Attributes
boolean	
character	Succ,Pred,Pos,Val
enumeration	
First,Last,Succ,Pred,Pos,Val	
float	Digits,Small,Large
integer	First,Last

Use of Scalar Types

```
Found : boolean;
Flag  : boolean := True;

Letter1 : character
Letter2 : character := 'A';
```

Text_IO contains a generic package called Enumeration_IO, which can be instantiated to provide IO routines for enumeration types.

```
with Text_IO; use Text_IO;
procedure Enum is
   type Days is (Mon,Tue,Wed,Thu,Fri,
                          Sat,Sun);
                 subtype WeekEnd is Days
range Sat..Sun;
   package Days_IO is new
         Enumeration_IO(Days);
   package WeekEnd_IO is new
      Enumeration_IO(WeekEnd);
   Today      : Days;
   Tomorrow : WeekEnd;
begin
   Today      := Fri;
   Tomorrow  := Sat;
   Put(Today);     -- write current
day
   Put(Tomorrow); -- write next day
end Enums;


Tax_Rate : float := 0.12;
Pi : constant float := 3.14

Count : integer;
StartNum : integer := 1;
```

Unary Operators
- Arithmetic Operators

Operation	Operator
absolute value	abs
unary plus	+
unary minus	-

- Boolean Operators

Operation	Operator
not	not

Binary Operators
- Arithmetic Operators

Operation	Operator
exponentiation	**
multiplication	*
division	/
modulus	mod
remainder	rem
addition	+
subtraction	-

- Boolean Operators*

Operation	Operator
and	and
or	or
exclusive or	xor
not	not

*When different Boolean operators are mixed in the same expression, parentheses are required for clarity. Thus the expression:

```
   (i = j) or (k = 1) and (k = i)
```

is invalid and must be written:

```
   ((i = j) or (k = 1)) and (k = i)
```

When using the binary operators and, or, and xor, both operands are always evaluated. Short circuiting an evaluation must be done explicitly using the and then and or else operators. For example:

```
N: integer := 0;
I,J: integer := 1;
...
if I = J or else J / N = 0 then ...
```

short circuits before J / N can be evaluated thus preventing a divide by 0 run-time error

```
if (TestCount > 0) and then
      (TestTotal/TestCount > 65) then

   Put_Line("Passing grade");

else

   Put_line("Failing grade");

end if;
```

short circuits before TestTotal/TestCount can be evaluated and cause a run-time error.

- Relational Operators

Operation	Operator
equality	=
inequality	/=
less than	<
less than or equal	<=
greater than	>
greater than or equal	>=

- Membership Test Operators

in, not in are used to determine if the value of an expression falls within a given range or subtype. For example:

```
Num : integer;
if (Num in 1..100) then
   ...
```

given the subtype declaration:

```
subtype SmallNums is integer range
1..100;
```

the condition above could be expressed as:

```
if (Num in SmallNums) then ...
```

Operator Precedence

highest precedence	**,abs,not
multiplying	*,/,mod,rem
unary adding	+,-
binary adding	+,-,&
relational/membership	=,/=,<,<=,>,>=, in,not in
logical	and,or,xor,and then, or else

Type Conversions

The data type names integer and float can be used as function names to convert numeric values of one type to the other. Float values are rounded to the nearest integer when converted to integer. For example:

integer(2.4)	has value 2
integer(2.5)	has value 3
float(32/5)	has value 6.0

Structured Data Types

Type	Attributes
array	First,Last,Range,Length,First(n)*, Last(n),Range(n),Length(n)
record	Constrained,First_Bit,Last_Bit, Position

* n = the nth array in a multidimensional array.

Use of Structured Types

Arrays

Constrained

```
type Line is array(1..80) of
character;
type List is array(1..5) of integer;

A,B : Line;
Matrix : array(1..4,1..6) of float;
Vector : List;

Vector(4) := 15;
Matrix(2,3) := 4.7;
A(4) := B(7);              single
elements
A := B;                    whole arrays
A(30..39) := B(50..59); 10 element
slices
A := (1..80 => ' ');
B := (1..5 => 'A', others=> ' ');
```

Unconstrained

```
type Vector is array(integer range <>)
of   integer;
type Char_Count is array(character
range   <>) of integer;
subtype : Thousand is integer range
1..1000;

Ten : Vector(1..10);
Kilo : Vector(Thousand);

Ten(5) := Ten(2) + Kilo(900);
```

Use of Structured Types continues on back side =====>

Strings

```
File_Name : string(1..20);
ErrMsg    : constant string := error
-         file not found";
Line       : string(1..80);
```

Ada does not have variable length strings, necessitating extensive use of array slices. For example:

```
Put("Please type your name: ");
Get_Line(Line,Length);
Line(1..Length+6) := "Hello " &
Line(1..Length);
Put(Line(1..Length+6)); New_Line;
```

Records

Non-Discriminant Records

```
type DigitString is array(1..9) of
       character range '0'..'9';

type PartInfo is
     record
       PartNum : DigitString;
       Description : string(1..50);
       Price : float;
       InStock : integer := 0;
     end record;

type DayInfo is
     record
       Month: integer range 1..12;
       Day: integer range 1..31;
       Year: integer range 0..3000;
     end record;

type EmployeeData is
     record
       Name: string(1..20);
       BirthDate: DayInfo;
       Address: string(1..30);
     end record;

Part1, Part2 : PartInfo;
EmployeeList: array(1..200) of
                          EmployeeData;
Manager: EmployeeData;

Part1 := Part2;
Employee(3) := Manager;
if (Employee(i) = Employee(i+1) then
   Duplicate := True;
end if;

Part1.PartNumber := "000045005";
Part2.PartNumber(4) := '5';
EmployeeList(3).Name := "John Doe
";
Manager.BirthDate.Month := 9;
```

Discriminant Records

```
type CreditLine (Limit : integer) is
     record
       Ceiling : integer := Limit;
       Balance : float := 0;
     end record;

type Days is (Mon,Tue,Wed,
                    Thu,Fri,Sat,Sun);

type Meeting (Hour: integer := 1300,
              Day: Days := Fri) is
     record
       M_time  : integer := Hour;
       M_Day   : Days := Day;
       M_Place : string(1..30);
     end record;


type Buffer (Size : integer := 80) is
     record
       Contents : string(1..Size);
       Location : integer := 0;
     end record;

type Matrix is array(integer range <>,
       integer range <>) of     float;

type SqMatrix (n : integer) is
     record
       Mat : Matrix(1..n,1..n);
     end record;

NewAcct   : CreditLine(2000);
Interview : Meeting;
Lunch     : Meeting(1200,Wed);
Line      : Buffer;
ShortLine : Buffer(40);
LongLine  : Buffer(256);
Grid      : SqMatrix(5);
```

Variant Records

```
type EmployeeInfo (PayType : integer)
is
     record
       Name    : string(1..30);
       Address : string(1..40);
         case PayType is
           when 1 | 2 =>
             Wages : float;
             Hours : float;
             OTime : float;
           when 3..10 =>
             Rate  : float;
             Sales : float;
           when 11 =>
             Salary: float;
           when others =>
              null;
     end record;

Empl_1 : EmployeeInfo(1);
Empl_2 : EmployeeInfo(7);
Empl_3 : Employee(11);
```

Access Data Type (Pointers)

Type	Attribute
access	Storage_Size

Use of Access Types

```
type CharPtr is access Character;

type Vector is array(1..4) of integer;
type VectorPtr is access Vector;

Cpt      : CharPtr;
Vpt      : VecPtr;*
```

*Access variables are, by default, initialized to the access value null.

```
- allocating memory
Cpt := new character;
Vpt := new Vector;

- referencing access types
Cpt.all := 'x';
Vpt.all(3) := 4;*
```

*For arrays and records, selected components may be referenced without the ".all" notation. For example: Vpt(3) := 4;

Control Statements

Selection

```
if (Score > 60) then
   status := passing;
else
   status := failing;
end if;

case Score is
   when 90..100=>  Grade := 'A';
   when 80..89=>  Grade := 'B';
   when 70..79=>  Grade := 'C';
   when 60..69=>  Grade := 'D';
   when 0..59=>  Grade := 'F';
   when others=>  null;
end case;
```

Loop Statements

```
loop
   Count := Count + 1;
   Put(Count,3);
   exit when (Count = 10);
end loop;

while (Count < 10) loop
   Count := Count + 1;
   Put(Count,3);
end loop;

for Count in 1..10 loop
   Put(Count,3);
end loop;*
```

*The loop control variable must not be declared as a program variable. It is implicitly declared by its use in the loop construct, and its scope is restricted to the loop construct. Thus it redefines any identifier of the same name within its scope.

The Block Statement

```
declare
   Temp : integer := Num_1;
begin
   Num_1 := Num_2;
   Num_2 := Temp;
end;*
```

*the declarative part may be referenced only within the block.

Procedures/Functions

```
procedure Swap(a,b: in out integer) is
   Temp : integer := a;
begin
   a := b;
   b := Temp;
end Swap;

function Is_Digit(Char : in character)
                                return
boolean is
begin
   return (Char in '0'..'9');
end Is_Digit;
```

VIII RESOURCES

a. Software Engineering Bibliography

BOOKS

This bibliography consists of a list of several software engineering text books which we have found useful. Also included are some books related to particular software engineering issues discussed in the course. Recently, Wiley Interscience has published the Encyclopedia of Software Engineering, edited by John Marchiniak. It is a very useful resource for most topics discussed in the course. Following the list of texts, is a list of articles arranged by subject.

Abdel-Hamid, Tarek, et al,
Software Project Dynamics: An Integrated Approach, Prentice Hall, Inc., 1991.

Arthur, Lowell J.,
Software Evolution: The Software Maintenance Challenge, John Wiley & Sons, 1988.

Bell, Doug, et al,
Software Engineering: A Programming Approach, Prentice/Hall International, 1987.

Berzins, Valdis, et al.
Software Engineering with Abstractions, Addison-Wesley Publishing Co., 1991.

Boehm, Barry W.,
Software Engineering Economics, Prentice-Hall, Inc., 1981.

Booch, Grady,
Object Oriented Design with Applications, The Benjamin/Cummings Publishing Company, Inc., 1991.

Brooks, Jr., Frederick P.,
The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley Publishing Co., 1975.

Burch, John G.,
Systems Analysis, Design, and Implementation, Boyd & Fraser Publishing Co., 1992.

Conger, Sue,
The New Software Engineering, Wadsworth Publishing Company, 1994.

Fairley, Richard,
Software Engineering Concepts, McGraw-Hill Book Co., 1985.

Fertuck, Len,
Systems Analysis and Design with Case Tools, Wm. C. Brown Publishers, 1992.

Gehani, N., et al,
Software Specification Techniques, Addison-Wesley Publishing Co., 1986.

Ghezzi, Carlo, et al.
Fundamentals of Software Engineering, Prentice Hall, Inc., 1991.

Gilb, Tom, et al.
Principles of Software Engineering Management, Addison-Wesley Publishing Co., 1988.

Higgins, David A.,
Data Structured Software Maintenance: The Warnier/Orr Approach, Dorset House Publishing Co., 1986.

Ince, D. C.,
Software Engineering, Van Nostrand Reinhold (International), 1989.

Jacobson, Ivar,
Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley Publishing Co., 1992.

Jones, Gregory W.,
Software Engineering, John Wiley & Sons, 1990.

Keller, Robert
The Practice of Structured Analysis: Exploding Myths, Yourdon Press, 1983.

Keyes, Jessica,
Software Engineering Productivity Handbook, Windcrest/McGraw-Hill, 1993.

Kowal, James A.,
Analyzing Systems, Prentice Hall, 1988.

Lamb, David Alex,
Software Engineering: Planning for Change, Prentice Hall, 1988.

Londeix, Bernard,
Cost Estimation for Software Development, Addison-Wesley Publishing Co., 1987.

Macro, Allen,
Software Engineering: Concepts and Management, Prentice Hall International, 1990.

Martin, James, et al,
Software Maintenance: The Problem and Its Solutions, Prentice-Hall, Inc., 1983.

Mynatt, Barbee Teasley,
Software Engineering with Student Project Guidance, Prentice Hall, Inc. 1990.

Pfleeger, Shari Lawrence,
Software Engineering: The Production of Quality Software, Second Edition, Macmillan Publishing Co., 1991.

Pfleeger, Shari Lawrence,
Software Engineering: The Production of Quality Software, Macmillan Publishing Co., 1987.

Powers, Michael J., et al,
Structured Systems Development: Analysis, Design, Implementation, 2d Edition, Boyd & Fraser Publishing, 1990.

Pressman, Roger S.,
Software Engineering: A Beginner's Guide, McGraw-Hill Book Co., 1988.

Pressman, Roger S.,
Software Engineering: A Practitioner's Approach, Third Edition, McGraw-Hill, Inc., 1992.

Price, Jonathan, et al,
How to Communicate Technical Information: A Handbook of Software and Hardware Documentation, The Benjamin/Cummings Publishing Co., Inc., 1993.

Schach, Stephen R.,
Software Engineering, Second Edition, Richard D. Irwin, Inc., and Aksen Associates, Inc., 1993.

Shooman, Martin L.,

Software Engineering: Design, Reliability and Management, Mc-Graw Hill Book Co., 1983.

Sigwart, Charles D., et al,
Software Engineering: A Project Oriented Approach, Franklin, Beedle & Associates, Inc., 1990.

Sommerville, Ian,
Software Engineering, Fourth Edition, Addison-Wesley Publishing Co., 1992.

Turner, Ray,
Software Engineering Methodology, Reston Publishing Co., Inc., 1984.

van Vliet, Hans,
Software Engineering: Principles and Practice, John Wiley & Sons Ltd., 1993.

von Mayrhauser, Anneliese,
Software Engineering: Methods and Management, Academic Press, Inc., 1990.

Weinberg, Gerald M.,
Quality Software Management: Volume 1 Systems Thinking, Dorset House Publishing, 1992.

Wiener, Richard, et al,
Software Engineering with Modula-2 and Ada, John Wiley & Sons, 1984.

Woodcock, Jim, et al,
Software Engineering Mathematics, Addison-Wesley Publishing Co., 1988.

ARTICLES

Background material section

Software Development Process subsection

Hall, A. "Is software engineering?" *Software Engineering Education* 1992, pp. 5-7.

Davis, A., E. Bersoff, et al. "A strategy for comparing alternative software development life cycle models." *IEEE Transactions on Software Engineering*, SE-14:10, October 1988, pp. 1453-1461.

Myers, W. "Allow plenty of time for larger-scale software." *IEEE Software*, July 1989, pp.92-99.

Brooks, F. "No silver bullet." *IEEE Computer*, April 1987, pp. 10-19.

Lewis, T., and P. Oman. "The challenge of software development." *IEEE Computer*, November 1990, pp. 9-12.

Software Engineering Environment subsection

Dart, S., R. Ellison, et al. "Software development environments." *IEEE Computer*, November 1987, pp. 18-28.

Kernighan B., and J. Mashey. "The UNIX programming environment." *IEEE Computer*, April 1981, pp. 12-22.

Software Engineering Techniques section

Bergland, G. "A guided tour of program design methodologies." *IEEE Software*, October 1981, pp. 13-37.

Schonber, E., M. Gerhardt, et al. "A Technical Tour of Ada." *Communications of the ACM*, 35:11, November 1992, pp. 43-52.

Object-Oriented Design Section

Nerson, J-M. "Applying object-oriented analysis and design." *Communications of the ACM*, 35:9, September 1992, pp. 63-74.

Berard, E. "Understanding object-oriented technology." *Essays on Object-Oriented*

Software Engineering, Vol. I, pp. 1-10, Prentice-Hall, 1993.

Berard, E. "Motivations for an object-oriented approach to software engineering." *Essays on Object-Oriented Software Engineering*, Vol. I, pp. 13-28, Prentice-Hall, 1993.

Tsichritzis, D. "Object-oriented development for open systems." *Information Processing 89*, Elsevier Science Publishers, 1989, pp. 1033-1040.

Object-oriented analysis, design and implementations. An example. (Lecture Notes)

Formal Methods of Software Development Section

Gerhart, S. "Application of formal methods: Developing virtuoso software." *IEEE Software*, 7:5, September 1990, pp. 7-10.

Cooke, J. "Formal methods --- mathematics, theory, recipes, or what?" *The Computer*, 35:5, 1992.

Hall, A. "Seven myths of formal methods. *IEEE Software*, September 1990, pp. 11-18.

Galton, A. "Classical Logic: A crash course for beginners. *The Computer*, 35:5, pp.424-430.

Spivey, J. "An introduction to Z and formal specifications." *Software Engineering Journal*, January 1989, pp. 40-50.

Future Directions section

Musa, J. "Software engineering: The future of a profession." *IEEE Software*, January 1985, pp. 55-62.

Shaw, M. "Prospects for an engineering discipline of software." *IEEE Software*, November 1990, pp. 15-24.

Cox, B. "Planning the software industrial revolution." *IEEE Software*, 7:6, November 1990, pp. 25-32.

Tsichritzis, D., and S. Gibbs. "From Custom-made to Pre-a-Porter software." *Object Management*, University of Geneva, 1990.

Nierstrasz, O., S. Gibbs, et al. "Component-oriented software development." *Communications of the ACM*, 35:9, September 1992, pp. 160-165.

Korson, T., and V. Vaishnavi. "Managing emerging software technologies: A technology transfer framework." *Communications of the ACM*, 35:9, September 1992, pp. 101-111.

Humphrey, W. "Contracting for software." *Managing the Software Process*, Addison-Wesley, 1989.

Berztiss, A. "Engineering principles and software engineering." *Software Engineering Education*, 1992, pp. 437.
END OF LIST

Berzins, V., and Luqi. *Software Engineering with Abstractions.* Addison-Wesley, 1991.

Byrne, W.E. *Software Design Techniques for Large Ada Systems.* Digital Press, 1991. 314p.

VII RESOURCES

b. Ada Bibliography

There are two sources of the annotations for the Ada books below. One source is the Ada Books list published by the Ada Information Clearinghouse. These entries are indicated by an "*" after the entry. The other source is from a list compiled by Mike Feldman, education director of SIGAda.

Andrews, E., editors. *Concurrent Programming with Ada*. Benjamin-Cummings, 1993. *

Atkinson, C., et al. *Ada for Distributed Systems.* (Ada Companion Series) Cambridge University Press, 1988. 147p.
Describes the final report of the Distributed Ada Demonstrated (DIADEM) project, which studied the problems and developed solutions for using Ada to program real-time, distributed control systems. Demonstrates new techniques for controlling such systems from a distributed Ada program. *

Ausnit, C.N., et al. *Ada in Practice*. (Professional Computing Series) Springer-Verlag, 1985. 195p.
Identifies and resolves issues related to Ada usage and promotes effective use of Ada in general programming, design practice, and in embedded computer systems. Contains 15 case studies that cover five general areas of the Ada language: naming conventions, types, coding paradigms, exceptions, and program structure.*

Barnes, J.G.P. *Programming in Ada Plus an Overview of Ada 9X*. 4th edition. Addison-Wesley, 1994. 622p.
The fourth edition, while remaining focused on the current ANSI 83 standard, reflects the imminent Ada 9X standard in three ways: all features of Ada that will be affected by the Ada 9X standard are highlighted with icons and their design rationale described in detail; a full chapter on Ada 9X provides a tutorial and summary of the most important changes, including the increased support for object-oriented programming, the introduction of a hierarchical library structure and the inclusion of protected objects; full details of the syntax changes are provided in the appendices for easy reference. *

Barnes, J. *Programming in Ada*. 3rd edition. Addison-Wesley, 1989. 494p.
Discusses Ada using a tutorial style with numerous examples and exercises. Assumes readers have some knowledge of the principles of programming. Covers the following: Ada concepts, lexical style, scaler types, control structures, composite type, subprogram, overall structures, private types, exceptions, advanced types, numerics types, generics, tasking, external interfaces. *

Barnes' work has been one of the most popular "Ada books." Some students find it hard to see how the pieces fit together from Barnes' often fragmentary examples; it is difficult to find complete, fully-worked out, compilable programs. A version is available with the entire Ada Language Reference Manual bound in as an appendix.

Barnes, J. *Programming in Ada.* 2nd edition. Addison-Wesley, 1983.*

Ben-Ari, M. *Principles of Concurrent and Distributed Programming.* Prentice-Hall 1990. (OS/concurrency)

In my opinion, this is the best introduction to concurrency on the market. Ada notation is used for everything, but the focus is on concurrency and not on Ada constructs per se. I liked the CoPascal notation of the first edition better, but this book is still great. A software disk is promised in the preface; I had to work quite hard to get it from the publisher, which finally had to express-ship it from England. The software comes with a tiny Ada-ish interpreter, complete with Pascal source code, adapted from Wirth's Pascal/S via CoPascal. There are also some real Ada programs, most of which I've tested and found correct and portable.

Booch, G. *Software Engineering with Ada.* 3rd ed. Benjamin-Cummings, 1994.

Introduces Ada from a software engineering vantage. Addresses the issues of building complex systems. Includes new features in this second version: a more thorough introduction to Ada syntax and semantics, an updated section on object-oriented techniques to reflect the current state of knowledge and improved examples that illustrate good Ada style for production systems development. *

Booch, G. *Software Engineering with Ada.* (2nd edition) Benjamin Cummings 1987.

Another of the classical "Ada books." Introduces Booch's OOD ideas. Not for use to introduce Ada to novices, in my opinion; there are some nice fully-worked case studies but they begin too far into the book, after long sections on design, philosophy, and language elements. The earlier chapters contain too much fragmentary code, a common flaw in books that follow the LRM order.

Booch, G. *Object-Oriented Design, with Applications.* Benjamin Cummings, 1991.

This is a good comparative introduction to the "object-oriented (OO)" concept. The first half gives a balanced presentation of the issues in OO Design; the second half gives nontrivial examples from Ada, Smalltalk, C++, CLOS, and Object Pascal. The author tries to sort out the difference between object-based (weak inheritance, like Ada) and object-oriented (like C++) languages. My only real complaint is that Booch should have worked out at least some of his case studies using several different languages, to highlight the similarities and differences in the language structures. As it is, each case study is done in only a single language. The good news is that the book is remarkably free of the hyperbolic claims one sometimes finds in the OO literature. I think this book could be used successfully in a second-level comparative languages course.

Booch, Grady. *Software Components With Ada Structures, Tools, and Subsystems.*

Benjamin-Cummings, 1987. 635p.

Catalogs reusable software components and provides examples of Ada programming style. Presents a study of data structures and algorithms using Ada. *

This work is an encyclopedic presentation of data structure packages from Booch's OOD point of view. It is great for those who love taxonomies. It's not for the faint-hearted, because the volume of material can be overwhelming. It could serve as a text for an advanced data structures course, but it's thin in "big O" analysis and other algorithm-theory matters. The book is keyed to the (purchasable) Booch Components.

Bover,D. *Introduction to Ada*. Addison-Wesley, 1991. *

Bover, D.C.C., K.J. Maciunas, and M.J. Oudshoorn. *Ada: A First Course in Programming and Software Engineering*. Addison-Wesley, 1992.

This work is, to our knowledge, the first Ada book to emerge from Australia, from a group of authors with much collective experience in teaching Ada to first-year students. A number of interesting examples are presented, for example, an Othello game. The book is full of gentle humor, a definite advantage in a world of dry and serious texts. In the book's favor is the large number of complete programs. On the other hand, it is rather "European" in its terseness; American teachers may miss the pedagogical apparatus and "hand-holding" typically found in today's CS1 books. Generic units are hardly mentioned.

Bryan, D.L., and G. Mendal. *Exploring Ada*. Volume 1. Prentice-Hall, 1990. 411p.

Describes Ada's type model, statements, packages and subprograms. Includes programming features such as information hiding, facilities to model parallel tasks, data abstraction, and software reuse. *

This is an excellent study of some of the interesting nooks and crannies of Ada; it sometimes gets tricky and "language-lawyerly." Volume 2 takes up tasking, generics, exceptions, derived types, scope and visibility; Volume 1 covers everything else. The programs are short and narrowly focused on specific language issues. If you like Bryan's "Dear Ada" column in Ada Letters, you'll like this book. It is certainly not a book for beginners, but great fun for those who know Ada already and wish to explore.

Burns, A. *Concurrent Programming in Ada*. (Ada Companion Series) Cambridge University Press, 1985. 241p.

Reports on Ada tasking offering a detailed description and an assessment of the Ada language concerned with concurrent programming. *

I used this book for years in my concurrency course. It's roughly equivalent to Gehani's book, but its age is showing. Cambridge Press is not always easy to get books from, especially in the US.

Burns, A., and A. Wellings. *Real-Time Systems and Their Programming Languages.* Addison-Wesley, 1990. 575p.

Provides a study of real-time systems engineering, and describes and evaluates the programming languages used in this domain. Considers three programming languages in detail: Ada, Modula-2 and Occam2. *

Burns, A. *A Review of Ada Tasking.* (Lecture Notes in Computer Science Series, Volume 262) Springer-Verlag, 1987. *

Caverly, P., and P. Goldstein. *Introduction to Ada: A Top Down Approach for Programmers.* Brooks-Cole, 1986. 237p.

Organizes and emphasizes those features that distinguish Ada from other programming languages. Uses a cyclical approach to the treatment of many topics. Gives a brief history of the development of the Ada language. Introduces the I/O capabilities, procedures, character and numeric data types and subtypes, and the concept of an Ada program library. Discusses enumeration, array, record, and derived types and demonstrates how the package can be used to encapsulate data types. Explains access types and applications and the encapsulation of data objects in packages. Illustrates how finite-state machines can be represented by packages. Describes the essentials of tasking and deals with blocks and exceptions. Introduces the reader to private types, types with discriminates, and generic units. *

Chirlian, Paul M. *Introduction to Ada.* Weber Systems, 1985. 291p.

Provides a basic course in the Ada programming language. (Ada courses and/or self-study) *

Clark, Robert G. *Programming in Ada: A First Course.* Cambridge University Press, 1985. 215p.

Introduces the Ada programming language. Targets persons without previous experience in programming. Details how to design solutions on a computer. Concentrates on solving simple problems in the early sections: the later sections explore how packages can be used in constructing large reliable programs. Emphasizes central features such as data types, subprograms, packages, separate compilation, exceptions and files. ANSI/MIL-STD-1815A-1983 is referenced throughout the book. *

Cohen, N.C. *Ada as a Second Language.* McGraw-Hill, 1986. 838p.

Explains Ada to those who wish to acquire a reading and writing knowledge of the Ada language. Also a programming reference source. *

This book is a quite comprehensive exploration of Ada which follows the LRM in its presentation order. My graduate students like it because it is more detailed and complete than alternative texts. It's an excellent book for students who know their languages and want to study all of Ada. There are good discussions of "why's and wherefore's" and many long, fully-worked examples.

Cooling, J.E., *Introduction to Ada*. Chapman and Hall, 1993. 576p.

Introduction to Ada gives a comprehensive introduction to the subject, covering all the basic aspects of the language with reference to the particular strengths of Ada in real-time systems. It is written primarily for the novice programmer who lacks experience of modern high-level languages. *

Culwin. *Ada: A Developmental Approach*. Prentice-Hall, 1992.

Intended for use on courses which teach Ada as the first programming language. The book is designed to take the reader from the basic principles of programming to advanced techniques. This books provides a complete introduction to software development using the programming language, Ada. It is not only concerned with the production of Ada programs, but it is also an introduction to the process of implementation and testing. Features include: a carefully structured tutorial which includes software developments, design, testing, and production. *

This work introduces Ada along with a good first-year approach to software development methodology. Much attention is paid to program design, documentation, and testing. Enough material is present in data structures and algorithm analysis is present to carry a CS2 course. A drawback of the book is that the first third is quite "Pascal-like" in its presentation order: procedures, including nested ones, are presented rather early, and packages are deferred until nearly the middle of the book. This is certainly not a fatal flaw, but it will frustrate teachers wishing a more package-oriented presentation. The programs and solutions are apparently available from the author.

Dawes, J. *The Professional Programmers Guide to Ada*. Pittman Publishing, 1988. *

Delillo, N.J. *A First Course in Computer Science with Ada*. Richard D. Irwin, Inc., 1993. *

This book is a first in the Ada literature: a version comes with an Ada compiler, the AETech-IntegrAda version of Janus Ada. Author, publisher, and software supplier are to be commended for their courage in this. The book itself covers all the usual CS1 topics. In my opinion, the order of presentation is a bit too Pascal-like, with functions and procedures introduced in Chapter 5 (of 15) and no sign of packages (other than Text_IO) until Chapter 10. Unconstrained arrays and generics are, however, done nicely for this level, and Chapter 13 is entirely devoted to a single nontrivial case study, a statistical package. I wish there were more complete programs in the early chapters, to put the (otherwise good) discussion of control and data structures in better context.

Dorchak, S.F., and P.B. Rice. *Writing Readable Ada: A Case Study Approach*. Heath, 1989. 244p.

Contains a style guide, which gives suggestions for enhancing code readability; devotes a chapter to the discussion of concurrency, and advanced feature of modern programming languages; a fully coded Ada program, along with a sample run; a bibliography, which lists books and articles about Ada and software

engineering principles; two indexes, one devoted exclusively to references of case study modules and the other listing important topics and concepts. *

Elbert, T.F. *Embedded Programming in Ada.* Van Nostrand Reinhold, 1989. 523p.
Clarifies Ada for the practicing programmer and for the advanced engineering or computer science student. Assumes the reader has acquired a certain level of sophistication, general concepts normally found in introductory programming texts are not covered. Also, presumes the reader is familiar with operating systems and has a basic knowledge of some block-structured languages such as PL/I and Pascal. *

Feldman, M.B. *Data Structures with Ada.* Prentice Hall, 1985 (now distributed by Addison-Wesley). (CS2/data structures)
This book is a reasonable approximation to a modern CS2 book: "big O" analysis, linked lists, queues and stacks, graphs, trees, hash methods, and sorting, are all covered. The Ada is a bit old-fashioned, especially the lack of generics; the book was published before compilers could handle generics. The packages and other programs are available free from the author. The book is currently under revision with Addison-Wesley and should appear in 1993.

Feldman, M.B., and E.B. Koffman. *Ada Problem Solving & Program Design.* Addison-Wesley, 1991.
Designed to introduce the novice to a number of Ada features, such as subprograms, packages, operator overloading, enumeration types, and array-handling operations. Emphasizes throughout the book the principles of data abstraction, software engineering, problem solving, and program design. *

This work combines the successful material from Koffman's CS1 pedagogy with a software-engineering-oriented Ada presentation order. Packages are introduced early and emphasized heavily; chapters on abstract data types, unconstrained arrays, generics, recursion, and dynamic data structures appear later. The last five chapters, combined with some language-independent algorithm theory, can serve as the basis of a CS2 course. A diskette with all the fully-worked packages and examples (about 180) is included; the instructor's manual contains a diskette with project solutions.

Feuer, A.R., and N. Gehani. *Comparing & Assessing Programming Languages: Ada, C & Pascal.* (Software Series) Prentice-Hall, 1984. *

Fischer, C., and R. LeBlanc. *Crafting a Compiler.* Benjamin Cummings, 1988. (compilers)
This book uses Ada as its language of discourse and Ada/CS, a usefully large Ada subset, as the language being compiled. If you can get the "plain Pascal" tool software by ftp from the authors, you'll have a good translator-writing toolset. Skip the Turbo Pascal diskette version, which is missing too many pieces to be useful. I've used the book since it came out with both undergrad and graduate compiler courses; it embodies a good blend of theory and "how it's really done" coding.

Students like it. The authors have recently published a second version, which uses C as its coding language but retains Ada/CS as the language being compiled.

Fisher, D.A., editor. *Ada Language Reference Manual.* Gensoft Corp., 1986. *

Gauthier, M. *Ada: Un Apprentissage.* (in French). Dunod, 1989.
I found this an especially interesting, almost philosophical approach to Ada. The first section presents Ada in the context of more general language principles: types, genericity, reusability. The second section introduces testing and documentation concerns, as well as tasking; the third considers generics and variant records in the more general context of polymorphism. For mature Ada students in the French-speaking world, and others who can follow technical French, this book can serve as a different slant on the conventional presentations of the language. An English translation would be a real contribution to the Ada literature.

Gehani, N. *Ada: Concurrent Programming.* (2nd edition). Silicon Press, 1991.
This is a less formal, more Ada-oriented presentation of concurrency than the Ben-Ari work. I use both books in my concurrency course; its real strength is the large number of nontrivial, fully worked examples. Gehani offers a nice critique of the tasking model from the point of view of an OS person. The preface promises the availability of a software disk from the publisher.

Gehani, N. *Ada: An Advanced Introduction.* 2nd edition. Prentice-Hall, 1989. 280p.
Introduces advanced problem-solving in Ada. Emphasizes modular programming as good programming practice. *

I've always liked Gehani's literate writing style; he knows his languages and treats Ada in an interesting, mature, and balanced fashion. This book comes with a diskette sealed in the back of the book, which is advantageous because the book has numerous nontrivial, fully-worked examples.

Gehani, N. *Unix Ada Programming.* Prentice-Hall, 1987. 310p.
Focuses on the novel aspects of the Ada language and explains them by many examples written out in full. Examines the interesting differences between the Ada language and other programming languages. Also, notes the similarities between Ada, Pascal, C, PL/I, and Fortran. *

Gonzalez, D. *Ada Programmer's Handbook.* Benjamin-Cummings, 1991. *

Gonzalez, Dean W. *Ada Programmer's Handbook and Language Reference Manual.* Benjamin-Cummings, 1991. 200p. *

Habermann, A., and Dwayne E. Perry. *Ada for Experienced Programmers.* (Computer Science Series) Addison-Wesley, 1983. 480p.
Offers a comparative review of Ada and Pascal, using dual program examples to

illustrate software engineering techniques. *

Hibbard, Peter, et al. *Studies in Ada Style.* 2nd edition. Springer-Verlag, 1983. 101p.
Presents concepts of the abstractions embodied in Ada with five examples: a queue, a graph structure, a console driver, a table handler and a solution to Laplace's equation using multiple tasks. *

Jones, Do-While. *Ada in Action with Practical Programming Examples.* John Wiley & Sons, 1989.
Helps Ada programmers avoid common pitfalls and provides them with many reusable Ada routines. Discusses a variety of numeric considerations, user interfaces, utility routines, and software engineering and testing. Provides examples of Ada code. *

Katzan, H., Jr. *Invitation to Ada & the Ada Reference Manual.* Petrocelli, 1982. 429p.
Calls for the scientific computing community to adopt the Ada programming language. Part II is the Ada Reference Manual, 1980 version. *

Krieg-Brueckner, B., et al, editors. *Anna : A Language for Annotating Ada Programs.* (Lecture Notes in Computer Science Series, Volume 260) Springer-Verlag, 1987. *

Ledgard, Henry. *Ada: A First Introduction.* 2nd edition. Springer-Verlag, 1983. 130p.
Assumes that the reader has experience with some other higher order programming language. Outlines several key features of Ada; a treatment of the facilities-concept of data types, the basic statements in the language, subprograms, packages, and general program structure. *

Lomuto, N. *Problem-Solving Methods with Examples in Ada.* Prentice-Hall, 1987.(algorithms)
Inspired by Polya's classic How to Solve It, this book can make a nice addition to an Ada-oriented algorithms course. It makes too many assumptions about students' programming background to use as a CS1 book, and doesn't teach enough Ada to be an "Ada book." But it makes nice reading for students sophisticated enough to handle it. I'd classify it as similar to Bentley's Programming Pearls.

Luckham, David C., et al. *Programming with Specifications: An Introduction to Anna, a Language for Specifying Ada Programs.* (Texts and Monographs in Computer Science) Springer-Verlag, 1990. 416p.
Offers an in-depth look at ANNA, a form of the Ada language in which specially marked comments act as formal annotations about the program to which they are attached. *

Lyons, T.G. *Selecting an Ada Environment.* (Ada Companion Series) Cambridge University Press, 1986. 239p.

Provides an overview of the Ada Programming Support Environment (APSE). Covers six main issues in selecting an environment. Contains summaries of current approaches to likely problems, indications of deficiencies in existing knowledge, and checklists of questions to ask when considering a particular environment. *

Mayoh, B. *Problem Solving with Ada.* (Wiley Series in Computing.) Reproduction of 1982 edition *

Mendal, G., and D.L. Bryan, *Exploring Ada.* Volume 2. Prentice-Hall, 1992.
A method of presentation based on the Socratic method, provides coverage and the semantics of Ada. Discusses focused problems individually. The second volume expands on the larger issues dealing with Ada's more advanced features.*

Miller, N.E. and C.G. Petersen. *File Structures with Ada.* Benjamin/Cummings, 1990. (file structures)
Designed for a straightforward ACM-curriculum file structures course, this book succeeds at what it does. There are good discussions of ISAM and B-tree organizations. The software can be purchased a low cost from the authors; it seems to approximate in Ada all those C-based file packages advertised in programmer-oriented trade publications.

Nyberg, K.A., editor. *Annotated Ada Reference Manual.* 2nd edition. Grebyn Corp., 1991.
Contains the full text of ANSI/MIL-STD-1815A with inline annotations derived from the Ada Rapporteur Group of the International Organization for Standards responsible for maintaining the Ada language. *

This is the definitive work on Ada legalities, because it presents not only the full text of the LRM but also the official Ada Interpretations as prepared by the Ada Rapporteur Group of Working Group 9 of the International Organization for Standardization (ISO) and approved by that organization. These commentaries, interleaved with the LRM text, are promulgated by the Ada Joint Program Office, the American National Standards Institute (ANSI) agent for Ada, in the Ada Compiler Validation Suite (ACVC). They are thus binding upon compiler developers. I recommend this book as an essential volume in the library of every serious Ada enthusiast.

Pokrass, D., and G. Bray. *Understanding Ada: A Software Engineering Approach.* John Wiley and Sons, 1985. *

Saib, S., and R.E. Fritz. *Introduction to Programming in Ada.* HR&W, 1985 *

Saib, Sabina H., and R.E. Fritz. *Tutorial: The Ada Programming Language.* IEEE Computer Society, 1983. 538p.
Covers such topics as the history and current status of Ada; basic language; schedule for industry and DoD; preventing error; readable, maintainable, modular

systems; real-time features, portability; and environments for Ada. *

Savitch, W.J., et al. *Ada: An Introduction to the Art and Science of Programming.* Benjamin-Cummings, 1992.

Written specifically for the first programming course. It starts with variable declarations, simple arithmetic expressions, simplified input-output, and builds upward toward subprograms and packages. A chapter-by-chapter instructor's guide is also available, as is a program disk with more that 140 completed programs from the text. *

This is a straightforward adaptation of the well-known Savitch Pascal books. Ada is introduced in a Pascal-like order, with subtypes and packages introduced halfway through the book. This is purely a CS1 book. The final chapter covers dynamic data structures. There is minimal coverage of unconstrained array types; generics are introduced at the halfway point to explain Text_IO, then continued only in the final chapter. The authors intended this book to provide a painless transition to Ada for teachers of Pascal; one wishes they had taken advantage of the chance to show some of the interesting Ada concepts as well. Program examples from the text are available on disk, but only as part of the instructor's manual; a solutions disk is available for a fee from the authors.

Schneider, G.M., and S.C. Bruell. *Concepts in Data Structures and Software Development.* (with Ada Supplement by P. Texel). West, 1991. (CS2/data structures)

This work is not, strictly speaking, an Ada book; rather, it is a solid, language-independent approach to modern CS2. The language of discourse in the book is a Pascal-like ADT language rather like Modula-2 in style; some examples are coded in legal Pascal. The Ada supplement makes it usable in an Ada-based course, but the supplement is rather too terse (100 pages of large type) for my taste, and insufficiently well keyed to the book chapters. The supplement's effectiveness would be greatly enhanced by full translations to Ada of a large number of the book's examples.

Sebesta, R.W. *Concepts of Programming Languages.* (2nd ed.). Benjamin Cummings, 1993. (comparative languages)

If you've been around for a while, you might remember the late Mark Elson's 1975 book by the same title. This is similar: a concept-by-concept presentation, with -- in each chapter -- examples taken from several languages. I include this work in an "Ada list" because I like its nice, impartial coverage of Ada. I especially like the chapters on abstraction and exception handling. The book covers -- comparatively, of course -- most of the languages you'd like to see, including C, C++, Lisp, Smalltalk, etc., with nice historical chapters as well. The book is readable; my students like it. Our undergraduate and graduate courses both use it as a base text.

Shumate, K. *Understanding Ada.* 2nd edition, John Wiley & Sons. *

This would make a CS1 book if it included more overall pedagogy, independent of language constructs. Otherwise it is a nice introduction to Ada in fairly gentle steps. Lots of completely worked examples, right from the start. Doesn't follow the LRM order, which is great.

Shumate, K.C. *Understanding Ada: With Abstract Data Types.* 2nd edition. John Wiley & Sons, 1989. *

Skansholm, J. *Ada from the Beginning.* Addison-Wesley, 1988. 617p.

Describes the principles and concepts of programming in a logical and easy-to-understand sequence and discusses the important features of Ada (except parallel programming). Teaches the basic of writing computer programs. Emphasizes the fundamentals of good programming. Provides a grounding in the programming language Ada. Covers the following: programming designs, the basics of Ada, control statements, types, subprograms, data structures, packages, input/ouput, exceptions, dynamic data structures, files, and generic units. *

This book was one of the first to use Ada with CS1-style pedagogy. There are excellent sections on the idiosyncracies of interactive I/O (a problem in all languages), and a sufficient number of fully-worked examples to satisfy students. Generics, linked lists and recursion are covered at the end; there is no tasking coverage, but one would not expect this at CS1-level.

Smith. *Introduction to Programming Concepts and Methods with Ada.* McGraw-Hill, 1993. *

Stratford-Collins, M.J. *Ada: A Programmer's Conversion Course.* (Ellis Horwood Series in Computers & Their Applications) John Wiley & Sons, 1982. *

Tremblay, J., et al. *Programming in Ada.* McGraw-Hill, 1990. PG 486p.

Explains computer science concepts in an algorithmic framework, with a strong emphasis on problem solving and solution development. *

Volper, D., and M.D. Katz. *Introduction to Programming Using Ada.* Prentice-Hall, 1990. 650p.

Uses the spiral approach as the presentation methodology in this introductory course in Ada programming. *

This book uses a heavily "spiraled" approach to Ada, and is designed for a 2-semester course, covering nearly all of Ada eventually. There are lots of fully-coded examples, and good pedagogical sections on testing, coding style, etc. If you like spiraling, you'll like this. The down side is that you can't find all you need on a given subject in one place. It's at the other end of the scale from the "Ada books" that follow the Ada Language Reference Manual (LRM) order.

Wallace, Robert H. *Practitioner's Guide to Ada.* McGraw-Hill, 1986 373p.

Discusses the issues to be considered when making the transition to Ada, on selecting toolsets, and on using the language effectively. Covers the following: Ada as a language; Ada Oriented Development Environments; Ada oriented design methodologies; Ada policies and standards; Ada products and vendors; sources of Ada-related information; making the transition to Ada and other uses of Ada. *

Watt, D.A., B.A. Wichmann, et al. *Ada Language and Methodology*. Prentice-Hall, 1987.
This work presents some interesting programming projects, and the coverage of design and testing--at the level of a first-year student--is quite good. The first third of the book concentrates heavily on classical control and data structures, leaving exceptions, packages and even procedures until the "programming in the large" material in the second third. CS2 teachers will find too little concentration on algorithm analysis. On the other hand, tasking and machine-dependent programming are covered. Like the Shumate work, this book would make a suitable introduction to Ada for students with a semester or so of programming experience; it "jumps in" too quickly to satisfy the needs of neophytes and is not well-tailored to CS1 or CS2 needs.

Weiss, M.A. *Data Structures and Algorithms in Ada*. Benjamin/Cummings, 1993.
I'm taking a gamble here in reviewing a book I haven't seen, but I have perused the C version of this book and think it reaches its intended market -- data structures courses (CS7) -- rather well. There's a good mixture of theory and practice (ADT design, for example), and coverage of new topics like amortized algorithm analysis and splay trees. A book at this level should not pay too much attention to teaching a language; rather it should make good use of its language of discourse. The Ada version has not appeared yet at this writing; if it uses Ada as well as the C version uses C, the book is a winner at its level, and sorely needed in the Ada literature.

c. CASE Tools

There is a wide range of CASE tools available. The low end CASE tools are primarily drawing tools for a particular development notation. As the utility of the tools increase, they add functions such as data dictionaries, code generators and traceability.

Students like the tools which are code generators, but they are disappointed when they find out that most of them only generate the specification and not the bodies of packages.

In our class we used 3 tools: EasyCASE for structured development, OMTool for Rumbaugh's approach, and ObjectMaker to generate Ada.

The list of tools below gives the source of the tool, its functionality, its cost in 1994, ar the hardware/software environemt it in which it operates.

AD/M*ethod*
- Structured Solutions
- functions -
  - data and process modeling
  - business object/event modeling
  - technology modeling
  - project templates
  - enhancement & maintenance

OpenSELECT
- Meridian Software Systems, Inc.
  - Irvine, Calif.
- functions
  - Yourdon/Demarco methodologies
  - Ward Mellor and Hatley extensions
  - Jackson structured program charts
  - Chen ERD's
  - Constantine
  - State Transition
- DOS/Windows
- $795

EasyCASE Plus
- Evergreen CASE Tools
  - 8522 150 4th Avenue NE
  - Redmond Wash 98052
- functions
  - methodologies
    - Gane & Sarson
    - Yourdon/DeMarco

- Ward-Mellor/Harley
  - SSADM
  - Yourdon/Constantine
  - Marting
  - Chen
  - Bachman
  - IDEF1X
- diagram types
  - data flow
  - stat transition
  - structure
  - entity relationshop
  - data model
  - entity life history
  - logical data structure
- $495/$649/$795

MacAnalyst and MacDesigner
- Excel Software
- functions
  - structured analysis & design
  - real-time extensions
  - data modeling & screen prototyping
  - object-oriented analysis & design
  - data dictionary & requirement database
- Macintosh

SILVERRUN
- Computer Systems Advisers, Inc.
  - 50 Tice Blvd.
  - Woodcliff Lake, NJ 07675
- functions
  - reverse data engineering to ER models
  - graphical relation models from ER models
  - foreign keys, indexes and SQL schemas
- Windows / OS/2 / Macintosh

McCabe Tools
- McCabe Associates
- functions
  - generates unit and integration tests
  - verify executed test path on the flow graph
  - reverse engineering by abstracting system design
- UNIX / VMS / DOS

ForeSight
- Computer & Engineering Consultants
- $12500

PC-METRIC
- SET Laboratories
- PO Box 868
- Mulino, OR 97042
- Academic version $25.00

Teamwork
- Cadre Technologies Inc.
- 222 Richmond Street
- Providence, RI 02903
- Sparcstations, HP 300/400 workstations, IBM AIX, Ultrix, Apollo Domain, HP 700, and VMS

Paradigm Plus
- Protosoft, Inc.
- functions
  - open architecture
  - diagram editor
  - matrix editor
  - script language
  - automatic diagram generation
  - automatic digram leveling
  - Rumbaugh's OMT
  - Booch/Buhr OOD
  - HOOD method
  - EVB method
  - Yourdon/DeMarco/Gane/Sarson SASD
  - Chen/Bachman Entity relation
  - Customizable report/code generation
  - Code generation for C++, C, Ada, and others
- DOS/Windows, UNIX/Motif, OS/2, Sun, HP, RS6000

Corvision
- Cortex Corporation

Daisys
- S/Cubed, Inc.

Design/IDF & CPN
- Meta Software Corp.

EasyCase Professional
    Evergreen CASE Tools
    8522 150th 4th Ave NE
    Redmond Washington, 98052

ObjecTool
    Object International
    8140 N. MoPac 4-200
    Austin, TX 78759

    OOA-OOD
    Windows

ObjectMaker
    Mark V Systems
        Encino CA
    functions
        C code generator
        Ada code generator
        structured methods
        20 OO methods
    Windows / WindowsNT

OMT Tool
    General Electric Advanced Concepts Center
        640 Freedom Business Center
        PO Box 1561
        King of Prussia, PA 19406

POSE
    Computer Systems Advisers, Inc.
        50 Tice Boulevard
        Woodcliff Lake, NJ 07675
    functions
        DFD
        drawings of the diagrams